UMIT private universität für gesundheitswissenschaften,
medizinische informatik und technik

XML-basierte Kommunikation im IHE-XDS Kontext zur archetypen-basierten Suche

Bachelorarbeit von: Marco Schweitzer

Im Rahmen des Studiums Biomedizinische Informatik
an der Privaten Universität für
Gesundheitswissenschaften, Medizinische Informatik und Technik

Betreuer: Dr. Dipl.-Inf. Samrend Saboor

Hall in Tirol, 2012

Betreuerbestätigung

Hiermit bestätige ich, die vorliegende Abschlussarbeit betreut zu haben, und ich befürworte damit die Abgabe der von mir insgesamt positiv benoteten Arbeit.

_____

Datum und Unterschrift des Betreuers

_____

(Name des Betreuers in Blockbuchstaben)

Annahme durch das Studienmanagement

am: _____

von: _____

Zusammenfassung

Mit zunehmender Spezialisierung der medizinischen Berufsgruppen und der damit verbundenen Arbeitsteilung bei der Behandlung eines Patienten werden erhobene Dokumente für den jeweiligen Fall nur in der eigenen Institution zur Verfügung gestellt. Eine integrierte, medizinische Versorgung verlangt jedoch eine lückenlose Kommunikation unter allen beteiligten Berufsgruppen bei der Behandlung eines Patienten. Die elektronische Gesundheitsakte (EGA bzw. ELGA) schließt diese Lücke durch eine organisationsübergreifende Verteilung klinischer Dokumente. Durch die unüberschaubare Menge an Dokumenten, welche durch Vernetzung der einzelnen Einrichtungen entstehen, muss auch ein großer Zeitaufwand für das Recherchieren der Dokumente aufgebracht werden, was zu einem beachtlichen Problem in der ärztlichen Behandlung führt.

Das Projekt EHR-Arche der Privaten Universität für Gesundheitswissenschaften, medizinische Informatik und Technik (UMIT) und der Medizinische Universität Wien versucht diese Problem mit einem webbasierten Suchprogramm zu lösen, welches zu einer restriktiven Fragestellung eines Arztes alle relevante Suchergebnisse bzw. Dokumente einer EGA liefert. Das Projekt baut dabei auf dem IHE-XDS-Profil auf, welches um Akteure erweitert wird, die eine inhaltliche Suche von Dokumenten ermöglichen. Dies setzt voraus, dass die Dokumente vollstrukturiert nach dem Zwei-Schichten-Modell mit Referenzmodell und Archetypen vorliegen.

Der Document Consumer (Standard XDS-Akteur) wird im Projekt durch flexible Schnittstellen mit den neuen Akteuren verbunden. Sogenannte Infoitems, welche ein medizinisches Konzept (z.B. systolischer Blutdruck) zur Beschreibung der Archetypen darstellen, werden dabei im Document Consumer vom Benutzer zu einer gewünschten Suchanfrage zusammengestellt. Der neue Document Crawler-Akteur verarbeitet diese Suche und liefert die Ergebnisse (z.B. bestimmter Blutwert) mit Link zum relevanten Dokument an den Document Consumer. Der neue Archetype Repository Akteur (ATR) beinhaltet alle Infoitems und kann diese in einer strukturierten Form als XML anbieten.

In dieser Arbeit wurden die Schnittstellen des Document Consumers zu den neuen XDS-Akteuren geplant, implementiert und getestet. Dazu wurden folgende Teilziele abgearbeitet:

Teilziel 1: Zuerst wurde ein Konzept zur Erstellung einer Suchanfrage erstellt, welches zusammen mit den Schnittstellenspezifikationen der erweiterten Akteure als Grundlage der Implementierung dient. Dieses schreibt vor, dass die interne Erstellung einer Suchan-

frage über das Suchanfragen-Editor-Konzept ausschließlich als JAVA-Objekte und die Kommunikation zu den Akteuren über proprietäre XML-Nachrichten erfolgt.

Teilziel 2: Die aus dem Konzept resultierende Implementierung ermöglicht es, dass die als XML verfügbaren Infoitems aus dem ATR geladen und in eine JAVA-Struktur geparst werden können. Die Knoten dieser Struktur (Infoitems) können in einer eigenen Suchanfragen-Klasse beliebig kombiniert, parametrisiert (z.B. nur bestimmter Blutwert) und strukturiert werden. Aus dieser Klasse kann eine valide XML-Suchanfrage serialisiert werden, welche vom Document Crawler verarbeitet werden kann. Die Ergebnisse werden zusammen mit einem Link zum relevanten Dokument als XHTML an den Document Consumer retourniert und dort für den Benutzer dargestellt. In dieser Arbeit wurden für die Erstellung einer Suchanfrage ausschließlich die Funktionalitäten im Back-End implementiert, welche für die Schnittstellen relevant sind.

Teilziel 3: Sowohl Modultests der Schnittstellen, als auch ein Integrationstest im fertigen Document Consumer, wurden durchgeführt, um die Implementierung auf funktionale Anforderungen zu testen.

Abstract

Increasing specialisation in healthcare profession and the related division of work in treating patients lead to a restricted access in patient documents. Usually only the institution that has adjudged a document is able to admit it. However an integrated patient-centered care requires a close communication between all involved healthcare professions in treating a patient. The electronic health record (EHR, in Austria: ELGA) solves this problem by a cross-organisational distribution of healthcare documents. But this connection also causes unmanageable document overflow, which leads to a considerable problem in treatment because a lot of time is needed to look into all documents.

The project EHR-Arche of the University for Health Sciences, Medical Informatics and Technology (UMIT) and the Medical University of Vienna tries to solve this problem with a web-based search-tool that delivers all relevant documents out of an EHR in order to answer a restrictive question of a physician. The project is based on the IHE-XDS profile, which gets enhanced by two new actors that make it possible to find information in documents in regard to the content. The concept for a content-related search requires full structured documents, which are given by the dual-model approach with reference model and archetypes.

The Document Consumer (standard XDS actor) gets connected with flexible interfaces to the new actors. So called infoitems describe medical concepts in archetypes e.g. systolic blood pressure. They are used to form an arbitrary query in the document consumer. The new document crawler actor processes this query and returns the results (e.g. a certain blood result) to the consumer with a link to the relevant document. The new archetype repository actor (ATR) contains all archetypes with its infoitems and can offer them in a structured manner, so the document consumer is able to load these infoitems. In this thesis the interfaces from the document consumer to the new XDS-actors were planned, implemented and tested. Therefor following sub-goals were processed:

Sub-goal 1: First a concept for creating a query was made, which is used together with the interface specifications of the enhanced actors, to form the basis for the implementation. Thereby the concept specifies that proprietary XML-messages are used for communication between the actors, while JAVA-Objects are used in the document consumer internal.

Sub-goal 2: The concept-based implementation allows to load all infoitems, which are provided in structured XML-format from the ATR. It parses this XML-message and saves it in a tree-based JAVA-structure. The nodes of this structure (infoitems) can be free combined, constrained (e.g. only a certain blood result) and structured in the searchquery-class. This

class can then be serialised to a valid XML-query which can be processed by the document crawler. The results are returned in XHTML, so they can directly be presented in the document consumer front-end. This thesis describes only the back-end functions of creating a query that are relevant for the interfaces.

Sub-goal 3: Module tests for the interfaces, as well as an integration test in the final document consumer, are performed to test the implementation on functional offers.

Inhaltsverzeichnis

1. Einleitung 1
1.1. Gegenstand und Motivation 1
1.1.1 Hintergrund – Fachlich 1
1.1.2 Hintergrund – Projekt 2
1.2 Problemstellung 3
1.3 Zielsetzung 3
2. Grundlagen und Stand der Forschung 4
2.1 Elektronische Gesundheitsakte - EGA 4
2.1.1 ELGA 5
2.1.2 EpSOS 5
2.2 Integrating the Healthcare Enterprise - IHE 5
2.2.1 Das IHE Profil Cross-Enterprise Document Sharing - XDS 6
2.3 Zwei Schichten Modell – Referenzmodell, Archetypen 8
2.3.1 Ein-Schicht-Modell 9
2.3.2 Zwei-Schicht-Modell 10
2.4 Grundlagen zu XML 11
2.4.1 Markup Language 11
2.4.2 Extensible Markup Language - XML 11
2.5 Gütekriterien für das Wiederfinden von Information 13
2.6 Diabetes Mellitus 15
2.6.1 Diabetes Mellitus Typ 1 15
2.6.2 Diabetes Mellitus Typ 2 15
2.7 EHR ARCHE – Konzept für inhaltliche Suche 16
2.7.1 Erläuterung des Projektzieles 16
2.7.2 Das erweiterte XDS-Profil 17
3. Methoden und Vorgehensplanung 21
3.1 Vorgehensweise des Entwicklungsprozesses 21
3.2 Planung 22
3.2.1 Auswahl des XML-Parser-Frameworks 22
3.2.2 Verwendete Entwicklungstools 27
3.3 Zu Teilziel 1: Entwurf des Schnittstellenkonzeptes 27
3.3.1 Konzept für die Erweiterung des Document Consumers 27
3.3.2 Kommunikation der erweiterten XDS-Akteure 29

3.3.3 Konzepterstellung der Document Consumer Schnittstelle 35
3.4 Zu Teilziel 2: Implementierung 36
3.4.1 Schnittstelle zwischen ATR und Document Consumer 36
3.4.2 Schnittstelle zwischen Document Crawler und Document Consumer 39
3.5 Zu Teilziel 3: Test der Schnittstellen 41
3.5.1 Test der ATR Schnittstelle 41
3.5.2 Test der Document Crawler Schnittstelle 42
3.5.3 Integrationstest im Document Consumer 43
4. Ergebnisse 44
4.1 Zu Teilziel 1: Konzept für die Schnittstellen zu den erweiterten XDS-Akteuren 44
4.1.1 Konzept zur Erstellung einer Suchanfrage 44
4.1.2 Schnittstellenkonzept des Document Consumers 47
4.2 Zu Teilziel 2: Implementierung der Schnittstellen 48
4.2.1 Node-Klasse 49
4.2.2 DataContainer-Klasse 50
4.2.3 SearchQuery-Klasse 51
4.3 Zu Teilziel 3: Test der Implementierung 52
4.3.1 Test der ATR Schnittstelle 52
4.3.2 Test der Document Crawler Schnittstelle 54
4.3.3 Integrationstest im fertigen Document Consumer 58
5. Diskussion 64
5.1 Diskussion der Methoden 64
5.1.1 Vorgehensweise des Entwicklungsprozesses 64
5.1.2 Auswahl des Parserframeworks 64
5.2 Diskussion der Ergebnisse 65
5.2.1 Zu Teilziel 1: Konzept für die Schnittstellen zu den erweiterten XDS-Akteuren 65
5.2.2 Zu Teilziel 2: Implementierung der Schnittstellen 66
5.2.3 Zu Teilziel 3: Test der Implementierung 68
5.3 Bedeutung der Ergebnisse 68
6. Ausblick 70
7. Danksagung 71
Literaturverzeichnis 72
Abbildungsverzeichnis 75
Tabellenverzeichnis 77

1. Einleitung

1.1. Gegenstand und Motivation

1.1.1 Hintergrund – Fachlich

Dokumentation im Gesundheitswesen spielt schon seit Beginn ärztlicher Behandlungen eine entscheidende Rolle für eine funktionierende Gesundheitsversorgung. Laut Leiner et al. [1] sollen klinische Dokumente Auskunft über Feststellungen zum Behandlungsverlauf geben, die zu einem früheren Zeitpunkt, von anderen Personen oder anderen Stellen getroffen wurden. Der Arzt kann daraus eine adäquate Entscheidung für seine weitere Behandlung treffen. Heute sorgt die medizinische Arbeitsteilung für einen hohen Kommunikationsbedarf zwischen Ärzten, Pflegekräften, Laboratorien, etc. Da der Arzt nicht auf alle Dokumente der verschiedenen Abteilungen zugreifen kann, steht für die Behandlung nur eine lückenhafte Krankengeschichte zur Verfügung. Um eine bestmöglichste Entscheidung für Diagnose und Therapie treffen zu können, benötigt der Arzt jedoch möglichst die vollständige Krankengeschichte eines Patienten. Diese Informationslücke versucht man durch Vernetzung der einzelnen Gesundheitseinrichtungen und Verteilen der erhobenen Dokumente zu schließen [2]. Die sogenannte ELGA (elektronische Gesundheitsakte) soll in Österreich relevante Information in Form von Dokumenten zu einem Patienten organisationsübergreifend zur Verfügung stellen [3]. Für die Umsetzung der ELGA wird das „Cross-Enterprise Document Sharing“ (XDS) Profil, ein international anerkannter Standard der Initiative „Integrating the Healthcare Enterprise“ (IHE), verwendet, welches eine standardbasierte Spezifizierung für Registrierung und Verteilung von Dokumenten zwischen Gesundheitseinrichtungen und den Zugang zu den Dokumenten organisationsübergreifend bereitstellt [4].

Durch die Vernetzung der einzelnen Einrichtungen und durch die dauerhaft wachsende Dokumentation entsteht jedoch auch eine unüberschaubare Menge an Dokumenten eines Patienten. Laut Leiner et al. [1] werden pro Jahr ca. 6 Millionen einzelne Dokumente (Arztbriefe, Laborbefunde, etc.) an einem durchschnittlichen Universitätsklinikum erstellt.

Für den Arzt stellt dies ein beachtliches Problem dar, da viel Zeit für das Recherchieren von Dokumenten verloren geht. Dies führt auch zu einer mangelnden Akzeptanz einer organisationsübergreifenden Patientenakte, da die Ärzte negative Auswirkungen auf die ärztliche Behandlung befürchten [5, 6]. Die Herausforderung besteht darin, für eine restriktive Fragestellung des Arztes relevante Suchergebnisse bzw. Dokumente in diesem Informationsüberfluss schnell zu finden. Bei der Suche dürfen nicht zu viele unwichtige Dokumente gefunden werden, es dürfen aber auch keine relevanten Objekte fehlen.

1.1.2 Hintergrund – Projekt

Hier setzt das laufende Projekt „Archtetypen-basierte Elektronische Gesundheitsakte“, auch EHR-Arche bezeichnet, der Privaten Universität für Gesundheitswissenschaften, medizinische Informatik und Technik (UMIT) und der Medizinische Universität Wien an [7]. Ziel dieses Projektes ist es, dem Arzt genau die Dokumente zu liefern, nach denen er in einem von ihm definierten Suchkontext interessiert ist. Folgende Schritte werden dafür durchgeführt:

1. Es werden die jeweiligen Informationsbedürfnisse der ELGA Anwender anhand der Krankheit Diabetes Mellitus untersucht.
2. Es wird ein Konzept zur Bearbeitung dieser Bedürfnisse erstellt
3. Das erstellte Konzept wird in Testimplementierungen umgesetzt und evaluiert.

Die Informationsbedürfnisse der Ärzte wurden bereits erhoben und ausgewertet [8]. Als Basis für die Umsetzung des in Schritt 2 entwickelten Konzepts wurde das auch international verbreitete IHE XDS-Profil ausgewählt. XDS bietet u.a. die Möglichkeit, Dokumente anhand standardisierter Metadaten zu suchen. Bei den Metadaten handelt es sich um recht allgemeingültige Angaben, die das jeweilige Dokument rudimentär beschreiben bzw. klassifizieren, wie zum Beispiel Erstellungsdatum, Autor oder Art des Dokuments. Einfache Suchanfragen können ohne weiteres durch die Metadaten-basierte Suche abgehandelt werden. Hierbei wird ausschließlich anhand der Metadaten entschieden, ob Dokumente für die jeweilige Suche relevant sind. Dementsprechend stößt die Metadaten-basierte Suche dann an ihre Grenzen, wenn der Arzt an Fragestellungen interessiert ist, welche aus dem Inhalt der Dokumente zu beantworten sind. Diese Notwendigkeit ist aufgrund der Ergebnisse des ersten o.g. Projektschrittes gegeben. Eine grundlegende Voraussetzung für die inhaltliche Suche ist, dass die durchsuchten Dokumente vollstrukturiert vorliegen. Dies wird im Projekt anhand der Spezifizierungen EN/ISO 13606 [9], „Health Level Seven Clinical Document Architecture“ (HL7 CDA) [10] und openEHR [11] realisiert. Darüber hinaus

erfordert die inhaltliche Dokumentensuche die Erweiterung der XDS Architektur um Dienste bzw. Akteure, die die entsprechende inhaltliche Auswertung der Dokumente ausführen. Die Erweiterung besteht aus:

1. Document Crawler: Akteur für die inhaltliche Auswertung
2. Archetype-Repository: Sammlung einzelner Strukturelemente vollstrukturierter Elemente

Auf die beiden Akteure und das vollständige Konzept für die Erweiterung des XDS-Profils wird detailliert im Grundlagenteil der Arbeit eingegangen. Die Anbindung dieser neuen Akteure erfordert eine semantisch eindeutige und doch flexible Kommunikation zwischen den herkömmlichen XDS-Akteuren und den neu entwickelten. Das theoretische Konzept für eine passende Strukturierung der Kommunikation wurde bereits im Projekt erarbeitet. Es sieht die Verwendung von proprietären XML-Nachrichten vor. Allerdings fehlt eine adäquate Implementierung der Schnittstellen in den oben genannten neuen Akteuren, welche die Nachrichten verarbeiten (d.h. erzeugen und versenden sowie empfangen und auswerten) können.

1.2 Problemstellung

Problem: Es gibt noch keine adäquate Implementierung der Schnittstellen zwischen den herkömmlichen XDS-Akteuren und den erweiterten Akteuren.

1.3 Zielsetzung

Ziel: Es müssen die fehlenden Schnittstellen adäquat implementiert werden.

- Teilziel 1: Es muss ein Konzept für die Implementierung der Schnittstellen erarbeitet werden.
- Teilziel 2: Es muss die Implementierung anhand des Konzeptes umgesetzt werden.
- Teilziel 3: Es muss die Implementierung getestet und evaluiert werden.

2. Grundlagen und Stand der Forschung

2.1 Elektronische Gesundheitsakte - EGA

Im 21. Jahrhundert ist es längst nicht nur mehr die alleinige Aufgabe des Arztes für die Gesundheit eines Menschen zu sorgen. Auch sehr viele andere Berufssparten wie Pfleger, Laboratorien, Apotheken, Versicherungsträger, etc. sind ebenso am Behandlungsprozess eines Patienten beteiligt. Diese medizinische Zusammenarbeit ist heute unabdingbar für eine effiziente Gesundheitsversorgung. Eine große Rolle spielt dabei die medizinischen Dokumentation: Ziel der medizinischen Dokumentation ist es den berechtigen Personen alle relevanten (und nur die relevanten) Informationen zu einem oder mehreren Patienten und ihrer Behandlung zum richtigen Zeitpunkt, am richtigen Ort und in der richtigen Form zu geben [1]. Die Sammlung aller Behandlungsfälle eines Patienten in einer Versorgungsinstitution bezeichnet man als elektronische Patientenakte, kurz EPA [12]. Das Problem bisher ist, dass die verschiedenen Berufsgruppen meist nur Dokumente zum Patienten besitzen, welche in der eigenen Einrichtung erhoben wurden. Dies bezeichnet Haas [12] als institutionelle EPA, kurz iEPA. Für eine wesentlich präzisere, sichere und schnellere Diagnose benötigt der Arzt jedoch möglichst Auskunft über alle früheren Krankheiten des Patienten bzw. auch über seine bisherige Lebensweise. Auch für eine passende Therapie ist die komplette Krankengeschichte von Bedeutung: so können eventuelle Unverträglichkeiten, z.B. bei Medikamenten, ausgeschlossen werden. Diese Lücke, Dokumente zwischen verschiedenen Gesundheitseinrichtungen auszutauschen, versucht man anhand einer einrichtungsübergreifenden elektronischen Patientenakte, kurz eEPA zu schließen [12]. Beinhaltet die Patientenakte neben den gesamten Behandlungsfällen aller beteiligten Versorgungsinstitutionen auch zusätzliche Informationen aus paramedizinischen Bereichen und kann der Patient auch selbst Einträge in seiner Akte vornehmen, so spricht man von einer elektronischen Gesundheitsakte, kurz EGA[1].

Neben der gesetzlichen Situation, welche Sicherheit und Datenschutz der Dokumente beschreibt [13], gilt die semantische Interoperabilität als Schlüsselanforderung für eine funktionierende EGA Kommunikation [14]. Unter semantischer Interoperabilität versteht

[1] Im englischen auch als EHR (electronic health record) bezeichnet

man, dass die ausgetauschte Information zwischen Sender und Empfänger in exakt der selben Weise verstanden und interpretiert wird.

2.1.1 ELGA

Die österreichische Umsetzung einer EGA wird als ELGA (Elektronische Gesundheitsakte) bezeichnet. Sie befindet sich seit 2007 in Entwicklung und soll in Zukunft alle Dokumente zu einem Patienten beinhalten, welche in österreichischen Gesundheitseinrichtungen erhoben wurden [2]. Ziel ist es, die Entwicklung und Vernetzung bestehender und zukünftiger elektronischer Informations- und Dokumentationssysteme voranzutreiben. Die ELGA basiert auf dem IHE XDS-Profil. Die beinhalteten Dokumente besitzen eine einheitliche Struktur nach der „Health Level Seven Clinical Document Architecture“ (HL7 CDA) [10].

2.1.2 EpSOS

Auch in den anderen europäischen Ländern strebt man eine nationale Gesundheitsakte an [15]. Weiters versucht man seit Mitte 2008 auch EU-weit, die einzelnen elektronischen Gesundheitsakten miteinander zu verbinden. „Smart open systems for European patients“, kurz EpSOS ist ein Projekt, welches bis Ende 2013 Dokumentationssysteme der Gesundheitseinrichtungen verschiedener Länder verbindet, mit der Absicht, patientenbezogene, klinische Information EU-weit bereitzustellen [16]. Ziel ist es, die Qualität und Sicherheit der Gesundheitsversorgung eines Patienten auch im Ausland aufrecht zu erhalten.

2.2 Integrating the Healthcare Enterprise - IHE

„Integrating the Healthcare Enterprise“, kurz IHE, ist eine Initiative mit der Absicht maximale Interoperabilität zwischen IT-Systemen im Gesundheitswesen zu erzielen [17]. Aus Anwendungsfällen (use cases), welche in enger Zusammenarbeit zwischen Anwender und Hersteller erarbeitet werden, werden von IHE sogenannte Technical Frameworks erstellt (Abbildung 1). Diese Technical Frameworks sind Beschreibungen bzw. Richtlinien, wie bestehende etablierte technische Standards, wie zum Beispiel HL7 oder DICOM, in Unternehmen anzuwenden und zu implementieren sind. Die Ergebnisse der Technical Frameworks werden als Profile veröffentlicht. Ein Profil beschreibt die beteiligten Systeme und deren Interaktionen. Die Systeme selbst werden dabei als Akteure bezeichnet, während die Interaktion zwischen zwei Akteuren anhand einer Transaktion beschrieben wird. Jedes Profil ist einer IHE-Domäne zugeordnet. Als Domäne wird ein zusammengehöriges fachliches oder medizinisches Gebiet verstanden. Das im folgenden Abschnitt erläuterte

XDS-Profil ist zum Beispiel der „IT Infrastructure“ bzw. ITI Domäne untergeordnet, welche Profile für die Implementierung standardbasierter, interoperabler Lösungen zur Verbesserung des Informationsaustausches bereitstellt.

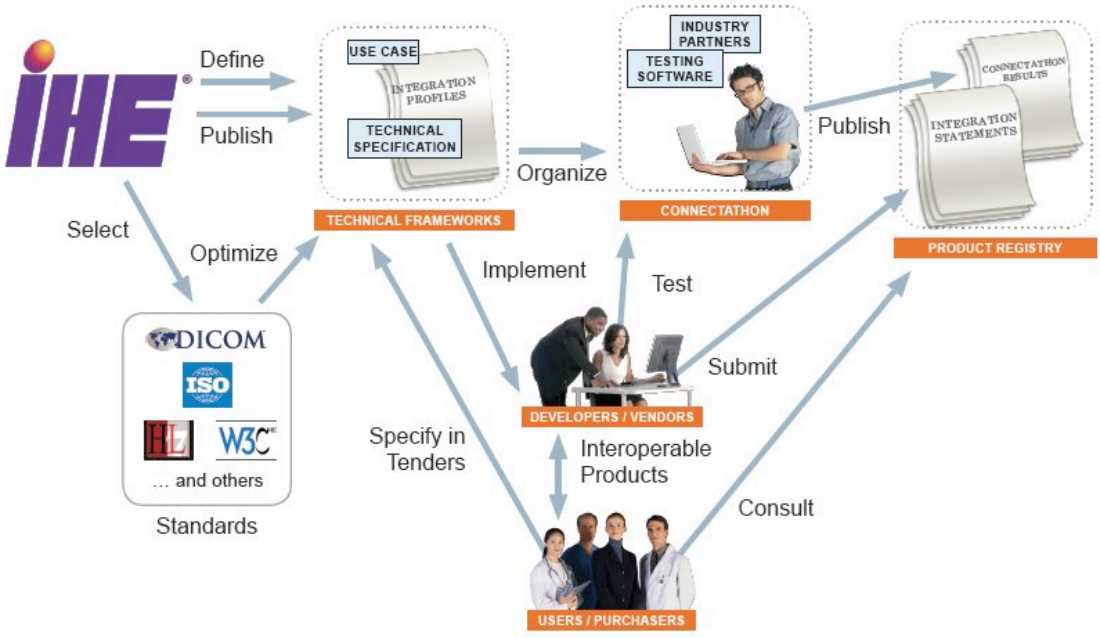

Abbildung 1: Ablauf des Entwicklungsprozesses eines Technical Frameworks [17].

Auf jährlichen „Connectatons“ wird die Interoperabilität der Neuentwicklungen von IHE auf Funktionalität und Praxistauglichkeit getestet und bewertet. Ziel von IHE ist es, die Technical Frameworks als Leitlinien für Softwarehersteller anzubieten, damit diese eine Software mit maximaler Interoperabilität für IT-Systeme im Gesundheitswesen entwickeln können.

2.2.1 Das IHE Profil Cross-Enterprise Document Sharing - XDS

„Cross-Enterprise Document Sharing“, kurz XDS, ist ein von IHE entwickeltes Profil, welches eine standardbasierte Spezifizierung für die Verteilung von Dokumenten zwischen Gesundheitsinstitutionen bereitstellt [4]. XDS beschreibt dafür Aktivitäten zur Speicherung und Registrierung von Dokumenten sowie den verteilten Zugriff auf registrierte Dokumente. Der Inhalt der Dokumente ist dabei nicht auf einfachen Text begrenzt, sondern kann auch als strukturiertes Textformat (z.B. HL7 CDA) oder Bild (DICOM) gespeichert werden. XDS verwendet das nach Haas definierte Prinzip der dezentralen Datenhaltung mit zentraler Metadatenhaltung [12]. Dabei werden die Dokumente lokal bei den jeweili-

gen Betrieben gespeichert, während das gemeinsam genutzte Serversystem die Metadaten und eine Verlinkung zu den lokalen Dokumenten enthält. Das Profil besteht aus folgenden Akteuren, welche durch die Transaktionen in Abbildung 2 verknüpft werden:

- Document Repository: ist verantwortlich für die persistente Speicherung und die gleichzeitige Registrierung der Dokumente in der Document Registry. Es bestimmt auch eine eindeutige ID für jedes Dokument, um von einem Document Consumer gefunden zu werden.
- Document Registry: beinhaltet die Liste mit den Metadaten jedes registrierten Dokuments mit dem Link zum Speicherort in dem Document Repository. Metadaten sind administrative Dokumentinformationen, wie zum Beispiel Autor, Erstellungsdatum und Dokumententyp. Die Document Registry dient zum aufschlüsseln der Suchanfrage eines Document Consumer Akteurs.
- Document Source: ist der Akteur, welcher neue Dokumente erstellt und zur Document Registry hinzufügt. Er ist verantwortlich für das Senden der Dokumente zum Document Repository und für das Bereitstellen der Metadaten für das Document Repository, welche anschließend für die Registrierung in der Document Registry verwendet werden.
- Document Consumer: bezeichnet die Anwender bzw. Anwendungssysteme, welche auf die Dokumente zugreifen möchten. Der Document Consumer stellt eine Suchanfrage an die Document Registry und empfängt die retournierten Dokumente eines oder mehrerer Document Repository Akteuren.
- Patient Identity Source: bietet eindeutige IDs jedes Patienten an, welche es erleichtern, einen Patienten in der Registry zu identifizieren. Dieser Akteur ist nicht vorgeschrieben, sondern kann optional hinzugefügt werden.
- Integrated Document Source/Repository: vereinigt den Document Source Akteur mit dem Document Repository Akteur. Somit muss keine „Provide&Register Document Set“ Transaktion definiert werden.

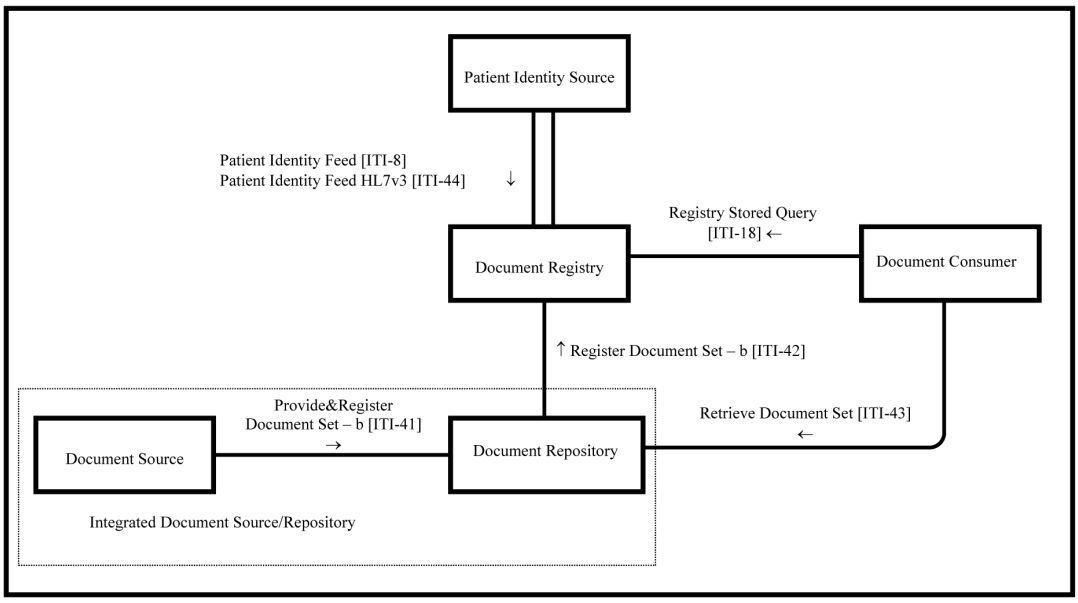

Abbildung 2: Darstellung des IHE XDS-Profils mit allen beteiligten Akteuren und Transaktionen [4].

2.3 Zwei Schichten Modell – Referenzmodell, Archetypen

Als Informationssystem bezeichnen Leiner et al [1] *„das Teilsystem eines Unternehmens (z.B. eines Krankenhauses oder einer anderen medizinischen Versorgungseinrichtung), das alle informationsverarbeitenden Prozesse und die an ihnen beteiligten menschlichen und maschinellen Handlungsträger in ihrer informationsverarbeitenden Rolle umfasst. Die informationsverarbeitenden Prozesse kann man sich als eine Menge formeller und informeller Tätigkeiten und Abläufe vorstellen, nach denen die Informationen gespeichert, verarbeitet und ausgetauscht werden.“*

Ein Gesundheitsinformationssystem (HIS[2]) erfüllt diese Aufgaben für die administrativen und klinischen Aspekte. Ein HIS verwaltet somit klinische Information und klinisches Wissen.

Als Information bezeichnet man Wissen, welches Auskunft über einen bestimmten Umstand bzw. über eine bestimmte Aktion gibt. „Der Patient XY hat einen arteriellen septalen Defekt , 1cm x 3,5cm“ wäre zum Beispiel eine medizinische Information [18]. Informationen gelten in der Regel nur für ein bestimmtes Objekt; das Beispiel kann also nicht auf andere Patienten übertragen werden.

[2] Health Information System

Leiner et al [1] beschreiben Wissen als *„ Information über den zu gegebener Zeit bestehenden Konsens im Bereich der Medizin und Gesundheitsversorgung, vor allem bezüglich der gültigen Terminologie, bestehender oder vermuteter zusammenhänge uns Gesetzmäßigkeiten, möglicher Interpretationen von Beobachtungen sowie empfehlenswerter Methoden und Handlungen. Im Gegensatz zur medizinischen Information im engeren Sinne bezieht sich medizinisches Wissen auf Begriffe statt auf konkrete Objekte."* „Das Septum teilt das menschliche Herz in eine linke und eine recht Herzkammer" beschreibt zum Beispiel medizinisches Wissen. Wie in der o.g. Beschreibung erwähnt, ist Wissen sehr allgemein, während sich Information sehr spezifisch, meist auf eine Person bezieht.

Information und Wissen können in einer Architektur auf verschiedene Arten miteinander verknüpft sein:

- Ein-Schicht-Modell (z.B. ENV13606)
- Zwei-Schichten-Modell (z.B. EN/ISO 13606 [9], HL7 CDA [10], openEHR [11])
- Mehr-Schichten-Modell

2.3.1 Ein-Schicht-Modell

Die verbreitetste Architektur beruht auf dem Ein-Schicht-Modell Ansatz, in welchem Information und Wissen zusammen als eine Einheit betrachtet werden. Softwareentwickler bauen somit das medizinische Fachwissen direkt in die Software ein. Vorteil ist, dass diese Methode im Verhältnis zu den anderen relativ leicht um zusetzen ist. Der Nachteil ergibt sich bei nähere Betrachtung: Die medizinische Versorgung bzw. das Gesundheitswesen ist eine Domäne, welche sich durch eine extrem große Menge an Wissen auszeichnet und sich auch ständig weiterentwickelt. SNOMED-CT beinhaltet z.B. mittlerweile ca. 800.000 Begriffe und über 1,3 Mio. Begriffsbeziehungen [19]. Durch die fortlaufende Forschung werden neue medizinische Erkenntnisse ergänzt, bzw. konventionelles Wissen angepasst oder erweitert. Im medizinischen Bereich muss das Wissen somit sowohl ein umfangreiches Gebiet abdecken können, als auch erweiterbar und leicht anpassbar sein. Das Ein-Schicht-Modell bietet hier kaum Möglichkeiten, das mit der Information eng verknüpfte Wissen zu erweitern bzw. zu ändern. Bei einer kleinen Änderung müsste die komplette Software neu entwickelt werden. Somit ist es offensichtlich, dass das Ein-Schicht-Modell für die medizinische Informationsverarbeitung nicht zukunftssicher verwendet werden kann.

2.3.2 Zwei-Schicht-Modell

Die Lösung bietet hier das Zwei-Schichten-Modell, welches auch als archetypenbasiertes Modell bezeichnet wird [20]. Hier werden Wissen und Information getrennt. Die Architektur besteht daher aus zwei Teilen:

- Einem generischen Teil für die Information → Informationsmodell, Referenzmodell
- Einem dynamischen Teil für das Wissen → Archetypen

Das Informationsmodell[3] (IM) beschreibt die allgemeinen Konzepte für eine medizinische Fachdomäne. Die EHR-Inhalte setzen sich aus dem Informationsmodell zusammen. Für ein zukunftssicheres System ist es jedoch wichtig, nicht alle EHR-Inhalte anhand des Informationsmodells zu beschreiben sondern nur jene, welche zu speziell sind, z.B. die Klassen welche sich im Laufe der Zeit nicht ändern. Die restlichen Inhalte werden aus Zusammenschlüssen von Instanzen des Informationsmodells beschrieben. Ein Archetyp beschreibt dabei welche Inhalte des Informationsmodells wie zusammengesetzt werden müssen, um einen bestimmten medizinischen Sachverhalt darzustellen.

Rinner [18] beschreibt den Zusammenhang zwischen Informationsmodell und Archetypen bildlich anhand von Lego©: Um ein Haus in Lego© zu bauen werden nur zwei verschiedene Legosteine benötigt: quadratische und rechteckige. Das Informationsmodell beinhaltet in diesem Zusammenhang die zwei Steinarten im übertragenen Sinne. Es gibt jedoch auch Objekte, welche nicht aus den zwei Steinen gebaut werden können, da sie zu speziell sind (wie z.B. Figuren, etc.). Diese werden somit auch als eigener Baustein erstellt und im Informationsmodell integriert. Werden die Inhalte des Informationsmodells richtig zusammengesetzt, kann man medizinische Sachverhalte abbilden. Das IM bietet jedoch nur die Bausteine dazu. Wie bei Lego© benötigt man noch einen Bauplan, wie die Steine zusammengesetzt werden müssen. Als Baupläne im Zwei-Schichten-Modell dienen die dynamischen Archetypen. Wissen wird somit nicht als starres Gebilde interpretiert, sondern kann beliebig erweitert bzw. geändert werden.

Das Zwei-Schicht-Modell vereinfacht die Erweiterung von Wissen in einem Informationssystem, der Entwicklungsprozess eines Zwei-Schichten-Modell ist aber von Grund auf aufwendiger als die Entwicklung eines Ein-Schicht-Modells.

[3] im EN/ISO 13606 Standard als Referenzmodell bezeichnet

2.4 Grundlagen zu XML

In diesem Abschnitt wird die Grundlage zu der Extensible Markup Language (XML) erläutert, die für das Erreichen der Ziele dieser Arbeit relevant sind.

2.4.1 Markup Language

Eine „Markup Language“ (Auszeichnungssprache) dient zur Beschreibung der Formatierung eines Textes. 1986 wurde die „Standard Generalized Markup Language“, kurz SGLM, als Standard eingeführt [21]. Sowohl das Design, wie z. B. fettgedruckt, kursiv, etc. als auch Gliederung eines Textes, z.B. Überschriften, Fußnoten, Absätze können mit einer Auszeichnungssprache klar definiert werden. Die bekannteste Auszeichnungssprache ist die „Hypertext Markup Language“, kurz HTML, welche zur Formatierung von Webseiten verwendet wird [22]. Der zu formatierende Text wird dabei in sogenannte Tags gestellt. Erkennbar sind sie an den spitzen Klammern (<,>). Schreibt man z.B. Test in ein HTML Dokument und ruft dieses in einem Browser auf, werden die Tags vom Browser verarbeitet und ein fettgedrucktes „**Test**“ wird im Browserfenster angezeigt[4]. Die Struktur kann anhand verschachtelter Tags aufgebaut werden.

2.4.2 Extensible Markup Language - XML

Die „Extensible Markup Language“, kurz XML, ist eine Auszeichnungssprache mit dem Ziel, einen Text zu strukturieren und ihn somit maschinenlesbar zu machen [23]. XML schreibt dabei anders als HTML keine Schlüsselwörter mit festgelegter Bedeutung vor, sondern legt lediglich den Aufbau und die Struktur fest. XML ist somit eine Metasprache zur Definition von Markup-Sprachen. Sie wurde 1998 vom „World Wide Web Consortium“ (W3C) entwickelt, um einen einfachen, flexiblen und benutzerfreundlichen Datenaustausch im Internet zu gewährleisten. Durch seine Flexibilität fand XML schnell eine weite Verbreitung (auch über reine Web-Anwendungen hinaus), so dass heute sehr viele Softwarelösungen mit XML oder XML-ähnlichen Strukturen für Datenaustausch und Speicherung arbeiten. Microsoft Word speichert z.B. ein erstelltes Dokument seit 2007 als XML, aber auch in medizinischen Bereichen hat sich XML etabliert: Der medizinische Kommunikationsstandard HL7 setzt ab der Version 3 ebenfalls auf XML Strukturen.

XML kann mit jedem üblichem Texteditor geschrieben werden. Die Dateiendung lautet „.xml“. Die Kopfzeile in einem XML-File gibt an, dass es sich um ein XML Dokument handelt und welche Zeichensatzkodierung verwendet wird. Danach folgt mit dem Wurzelement

[4] das steht in HTML für „bold“ und bedeutet „fettgedruckt“

der oberste Knoten, dem sich alle weiteren Elemente unterordnen. Ein Element in einem XML Dokument ist eine komplette Informationseinheit. Die Bezeichnung eines Elementes kann vom Benutzer frei gewählt werden. Es kann zwischen zwei Typen von Elementen unterschieden werden:

1. Elemente mit Inhalt: Das Element besteht aus einem öffnenden Tag, gefolgt vom eigentlichen Inhalt und einem schließenden Tag. Ein Beispiel aus Abbildung 2 ist das Element `<uhrzeit>11:05</uhrzeit>`.
2. Leeres Element: Das Element besteht aus einem leeren Tag (erkennbar an dem / nach der Tagbezeichnung. Beispiel: `<zustand notfall="false"/>`

Ein Element-Tag kann zusätzlich noch sogenannte Attribute beinhalten, welche Eigenschaften des Elements darstellen. Ein Attribut besteht aus einem Namen und einem in Anführungszeichen gestellten Wert. Es wird direkt in das öffnende Tag eines Elementes geschrieben, wie z.B. `notfall="false"`. Es ist auch möglich, mehrere Attribute zu einem Element zu definieren. Die einzige Voraussetzung ist, dass sich die Bezeichnungen unterscheiden.

Der Inhalt eines Elements kann aus Text oder weiteren Elementen bestehen, welche dem XML Format seine Struktur verleihen. Das Element „Patient“ aus Abbildung 2 beinhaltet z.B. die Elemente Uhrzeit, Kommentar und Zustand. Hält ein XML Dokument alle oben genannten Spezifikationen ein, wird es auch „wohlgeformt“ genannt. Wohlgeformtheit ist die Grundvorrausetzung für die Erstellung eines XML-Dokumentes.

```
<?xml version="1.0" encoding="UTF-8"?>
<patientenwarteliste datum="25.09.11">
    <patient name="Max Mustermann">
        <uhrzeit>11:05</uhrzeit>
        <kommentar>verkühlt</kommentar>
        <zustand notfall="false"/>
    </patient>
<patientenwarteliste>
```

Abbildung 3: Ein wohlgeformtes XML-Dokument

Um zu gewährleisten, dass nur bestimmte XML Dokumente von einer Anwendung als gültig angesehen werden, gibt es die Möglichkeit zusätzlich eine Grammatik zu definieren. Ein XML Dokument wird als gültig angesehen, wenn es wohlgeformt ist, mit einer Grammatik verknüpft wurde und das Dokument diese Grammatik berücksichtigt.

Dafür bedient man sich einer sogenannten Schemasprache. Die zwei bedeutendsten sind „Document Type Definition“ (DTD) und für komplexere Grammatiken auch „XML-Schema“. Beide geben im Grunde an, welche Tags in einem XML zwingend vorgeschrieben sind, welchen Inhalte in einem Element vorkommen dürfen (Text, Elemente oder beides in Kombination) und welche Attribute ein Element besitzt. Die Erläuterung der Konzepte, welche der DTD und XML-Schema zugrunde liegen, gehen über den Rahmen dieser Arbeit hinaus.

Durch diese unbeschränkte Einsetzbarkeit hat sich XML schnell als Standard für Austausch und Datenspeicherung erwiesen. Sowohl das Projekt EHR-Arche, als auch das IHE XDS-Profil verwenden XML-Strukturen für die Kommunikation zwischen den einzelnen Komponenten.

2.5 Gütekriterien für das Wiederfinden von Information

6 Millionen einzelne Dokumente (Arztbriefe, Laborbefunde, etc.) werden an einem durchschnittlichen Universitätsklinikum pro Jahr erstellt [1]. Diese Zahl verdeutlicht die ungeheure Informationsmenge im medizinischen Bereich. Durch die Einführung einer EGA wird diese Menge noch größer, was ein beachtliches Problem für das Suchen und Finden von Dokumenten darstellt.

Das Finden von Dokumenten in einer Menge muss in diesem Zusammenhang genauer erläutert werden: Eine Menge (z.B. alle Dokumente zu einem Patienten) besteht aus Objekten, welche für einen Suchkontext in Frage kommen und solche welche nicht in Frage kommen. Man kann auch sagen eine Menge besteht aus einer relevanten und einer nicht relevanten Teilmenge. Das Suchergebnis sollte im bestmöglichsten Fall alle relevanten Dokumente auflisten und keine der nicht relevanten Dokumente beinhalten. Dies führt zu einer aus der Statistik bekannten Beurteilung eines Klassifikators, welche sich anhand einer Kontingenztabelle verdeutlichen lässt [1].

Tabelle 1: Kontingenztabelle für das Wiederfinden von Information in einer Menge.

	relevant[5]	nicht relevant	Summe
gefunden	**Gefundene Objekte, welche auch relevant sind**	**Ballast**	Alle gefundene Objekte
nicht gefunden	**Verlust**		
Summe	Menge der gesamten relevanten Objekte		Alle Objekte in der Menge

Für die Bewertung einer Suche sind folgende Begriffe von Bedeutung [24]:

Die Sensitivität (auch Trefferquote oder Recall) gibt an, wie viele relevante Dokumente im Verhältnis zu den gesamten relevanten Dokumenten in der Menge gefunden wurden. Als Wahrscheinlichkeitsgleichung ausgedrückt:

$$Sensitivität = \frac{|\{relevante\ Dokumente\} \cap \{gefundene\ Dokumente\}|}{|\{relevante\ Dokumente\}|}$$

Die Genauigkeit (auch Prezision oder Relevanzrate) gibt an, wie viele der Treffer auch wirklich für die Fragestellung relevant sind.

$$Genauigkeit = \frac{|\{relevante\ Dokumente\} \cap \{gefundene\ Dokumente\}|}{|\{gefundene\ Dokumente\}|}$$

Tabelle 2 beschreibt eine Beispielsuche: ein Patient besitzt insgesamt 34 Dokumente. Der Arzt führt eine bestimmte Suche aus und bekommt 15 Dokumente aufgelistet. 10 davon wurden richtig gefunden, d.h. sie sind relevant, 5 dagegen sind nicht relevant. 3 relevante Dokumente wurden jedoch nicht gefunden.

Tabelle 2: Beispiel einer Kontingenztabelle von einer Dokumentensuche

	relevant	nicht relevant	gesamt
gefunden	**10**	**5**	15
nicht gefunden	**3**	**16**	19
gesamt	13	21	34

[5] im Bezug auf eine bestimmte Suchanfrage

Hier ist die Sensitivität = 10/13. Das bedeutet, dass ~77% der relevanten Dokumente gefunden wurden. Die Genauigkeit = 10/15. Dies bedeutet, dass genau 2/3 aller gefundenen Dokumente als relevant gelten.

Ziel einer guten Suche ist es, sowohl eine hohe Genauigkeit, als auch eine hohe Sensitivität zu erreichen, d.h. beide Werte sollten gegen 1 streben [24]. Eine gute Suche kann in der Praxis u.a. durch eine genauere Identifizierung der Dokumente erreicht werden. Jedes Dokument besitzt zwar Metadaten, wie Erstellungsdatum und Dokumententyp, daraus kann jedoch nicht auf den eigentlichen Inhalt des Dokuments geschlossen werden. Dies stellt in der medizinischen Domäne ein Hindernis dar, da die Dokumente in einem HIS meist unstrukturiert vorliegen und somit nicht semantisch maschinenlesbar sind. Genauigkeit und Sensitivität könnten somit durch ergänzendes semantisches Durchsuchen der Dokumente erheblich gesteigert werden. Technisch gesehen ist die Suche dann zwar aufwendiger, da jedes Dokument genauer analysiert werden muss, der Benutzer spart sich aber Zeit, die Information aus dem Dokument selbst herauszufiltern. Die gewünschte Information kann beim semantischen Durchsuchen direkt übernommen und exklusiv angezeigt werden.

2.6 Diabetes Mellitus

Diabetes Mellitus [25], auch Zuckerkrankheit genannt, bezeichnet eine Stoffwechselerkrankung, bei der durch fehlendes Insulin nur erschwert Glukose in die Zelle transportiert werden kann. Da Glukose als wichtiger Ausgangsstoff für die Energiegewinnung im Mitochondrium dient, kommt es somit zu einem Energiemangel in der Zelle und gleichzeitig zu einem erhöhten Glukosespiegel im Blut. Man unterscheidet zwei Arten:

2.6.1 Diabetes Mellitus Typ 1

Beim Diabetes Mellitus Typ 1 liegt ein absoluter Insulinmangel vor. Grund dafür ist, dass die Bauchspeicheldrüse effektiv zu wenig Insulin produziert. Vorwiegend betroffen vom Typ 1 sind Kinder, Jugendliche und junge Erwachsene.

2.6.2 Diabetes Mellitus Typ 2

Diabetes Mellitus Typ 2 bezeichnet einen relativen Insulinmangel. Durch Überernährung steigt der Insulinbedarf so stark an, dass die Insulin produzierenden B-Zellen im Pankreas

geschwächt sind und die Insulinproduktion nicht mehr ausreicht. 90% aller Diabetes Mellitus Fälle sind vom Typ 2. Hauptsächlich betroffen sind ältere, übergewichtige Menschen.

Ca. 50.000 Typ 1 Diabetiker und 500.000 Typ 2 Diabetiker leben allein in Österreich, womit hierzulande Diabetes auch gleichzeitig die Volkskrankheit Nr. 1 ist [26].

2.7 EHR ARCHE – Konzept für inhaltliche Suche

In diesem Abschnitt wird das vom Projektteam erarbeitete allgemeine Konzept zur Umsetzung der inhaltlichen Dokumentensuche erläutert.

2.7.1 Erläuterung des Projektzieles

Aus den Ergebnissen des 2. Projektschritts (Erhebung der Informationsbedürfnisse der ELGA Anwender) konnten insgesamt 446 medizinische Konzepte im Bezug auf Diabetes Mellitus erhoben werden [27]. Diese medizinischen Konzepte werden als Information-Items bzw. Infoitems bezeichnet. Ein Infoitem kann dabei ein einfacher Wert (z.B. „systolischer Blutdruck“) oder eine komplexere Darstellung (z.B. „Allergien“) sein. Die Informationsbedürfnisse mussten dann auf Archetypen abgebildet werden, damit diese für die Modellierung des EHR-Inhaltes verwendet werden können. Es wurden somit 128 Archetypen nach dem EN/ISO 13606 Standard erstellt, welche die 446 Infoitems enthalten bzw. repräsentieren [28].

Ziel des Projektes ist es, Suchanfragen auf EHR-Dokumente durchzuführen, welche sich auf Metadaten und/oder auf den Inhalt des Dokuments, also Archetypen, beziehen. Voraussetzung für eine inhaltliche Suche ist, dass die durchsuchten Dokumente vollstrukturiert vorliegen. Dies wird im Projekt EHR-Arche anhand des EN/ISO 13606 Standards gelöst. Dieser Standard spezifiziert eine Informations-Architektur zum Austausch von EHR-Inhalten nach dem Zwei-Schichten-Modell.

Es kann somit zwischen zwei Sucharten unterschieden werden:

1. Standard-Metadatensuche
2. Erweiterte archetypenbasierte Suche

Der Unterschied soll anhand zweier Beispiele verdeutlicht werden: Eine Suchanfrage könnte lauten: „Zeig mir alle Arztbriefe der letzten 6 Monate des Patienten XY“. Da das Erstellungsdatum in den Metadaten zum Dokument in der Document Registry enthalten

ist, lässt sich ein Arztbrief für den Zeitbereich ohne weiteres anhand der Standard-Metadatensuche finden.

Eine komplexere, den Inhalt betreffende Suchanfrage könnte lauten: „Zeig mir alle Dokumente der letzten 6 Monate, welche pathologische Blutzuckerwerte des Patienten XY aufweisen“. Hier ist es nötig Archetypen bei der Suche miteinzubeziehen. Man möchte die Dokumente des Patienten finden, die aufgrund der gesuchten Infoitems (im Beispiel „Blutglukose Messung“) den entsprechenden Archetypen zugeordnet werden können. In der erweiterten archetypenbasierten Suche kann nach beliebigen Infoitems bzw. Infoitem-Kombinationen gesucht werden. Zusätzlich können bei bestimmten Infoitems genaue Werteangaben bzw. Werteintervalle definiert werden, um noch spezifischere Suchen erstellen zu können. Die erweiterte archetypenbasierte Suche maximiert somit den Recall der Suche. Diese Art von Suche kann jedoch nicht mit den Standard XDS-Akteuren umgesetzt werden.

Tabelle 3: Vergleich von Metadatensuche und Archetypensuche

	Standard-Metadatensuche	**Erweiterte archetypenbasierte Suche**
Aufwand der Suchanfragenerstellung für den Benutzer	Verhältnismäßig gering	Verhältnismäßig höher
Vorliegende Dokumentenstruktur	Strukturiert (maschinenlesbar) oder unstrukturiert	Strukturiert (maschinenlesbar)
Recall	niedrig	hoch
XDS-Standard	ja	nein
Resultat der Suche	Relevante Dokumente	Relevante Ergebnisse mit Link zu Dokumenten

2.7.2 Das erweiterte XDS-Profil

Um die erweiterte archetypenbasierte Suche durchführen zu können, muss das XDS-Profil um Akteure erweitert werden, die eine inhaltliche Auswertung der XDS-Dokumente ermöglichen. Die neuen Akteure sollen dabei in keiner Weise die vorhandenen Strukturen des Profils beeinflussen bzw. ändern.

In Abbildung 4 sind die vom Projektteam geplanten Erweiterungen des XDS-Profils hellblau markiert. Die neuen Akteure umfassen:

1. Erweiterung des Document Consumers für Erstellung inhaltlicher Suchanfragen

2. Document Crawler
3. Archetype-Repository

Sie werden in den folgenden Abschnitten genau erläutert.

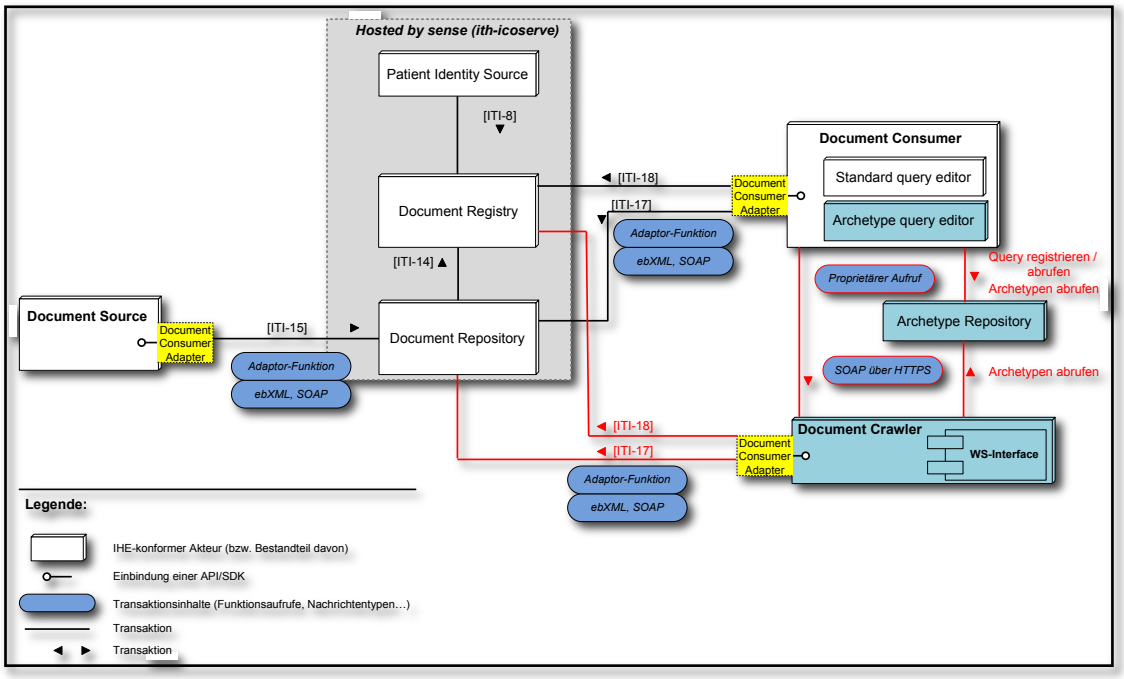

Abbildung 4: Architektur des erweiterten XDS-Profils. Das Standard XDS-Profil wird um einen Archetype Repository Akteur (für die Aufbewahrung der Archetypen) und einem Document Crawler Akteur (für die Verarbeitung inhaltlicher Suchanfragen) erweitert und mit den Standard-Akteuren verbunden.

2.7.2.1 Document Consumer

Der Document Consumer gehört zu den Standardakteuren des XDS-Profils und ist für die Erstellung einer Suchanfrage bzw. für die Ergebnisrepräsentation verantwortlich. Er kann in zwei Teile gegliedert werden:

1. Front-End (GUI)
2. Back-End (Funktion)

Das Front-End ist für die grafische Darstellung der Inhalte (sowohl für die Erstellung, als auch für die Ergebnisanzeige) verantwortlich und nimmt Benutzerinteraktionen entgegen. Das Back-End verarbeitet die Benutzereingaben zu einer Suchanfrage und schickt diese an die Akteure zum Ausführen der Suche weiter bzw. empfängt die Ergebnisse, welche anschließend für das Front-End aufbereitet werden.

Der Document Consumer Akteur wird im erweiterten XDS-Profil um eine Komponente erweitert, die es erlaubt, archetypenbasierte Suchanfragen zu erstellen. Transaktionen für die Kommunikation mit Archetype Repository (Abschnitt 2.7.2.3) und Document Crawler (Abschnitt 2.7.2.2) müssen erstellt werden. Die Erweiterung des Document Consumers muss in der Lage sein, Suchanfragen anhand festgelegter Archetypen zu erstellen, welche vom Benutzer beliebig zusammengestellt und zusätzlich parametrisiert werden können. Die genaue Funktionsweise dieser Komponente steht noch nicht fest. Die Schnittstellen des Document Consumers zu den neuen Akteuren werden in dieser Bachelorarbeit entwickelt. Die grafische Darstellung bzw. die Verarbeitung der Benutzerinteraktionen im Document Consumer ist Thema einer anderen Bachelorarbeit von Dominik Schweiger und wird somit hier nicht genauer erläutert.

2.7.2.2 Document Crawler

Der Document Crawler Akteur dient der Abarbeitung einer archetypenbasierten Suche in XDS-Dokumenten. Er beinhaltet die Logik für das optimierte durchsuchen von vollstrukturierten EN/ISO 13606 Dokumenten anhand der entsprechenden Archetypen. Eine Suchanfrage wird dabei in zwei Schritten durchgeführt:

1. Suchanfrage eines Document Consumers entgegen nehmen und entscheiden, ob es sich um eine Metadatensuche oder Archetypensuche handelt
2. Metadatensuche oder eine Archetypensuche durchführen

Die Metadatensuche wird vom Document Crawler direkt auf der Document Registry ausgeführt. Das funktioniert bei einer Archetypensuche nicht, da die Archetypen-IDs nicht in den Metadaten des Dokuments enthalten sind. Es ist jedoch möglich, anhand der Archetypen auf den Dokumenttyp zu schließen, da jeder Dokumenttyp auch gleichzeitig durch einen spezifischen Archetyp abgebildet wird (in ISO/13606 in der COMPOSITION, in CDA in ClinicalDocument). Die Archetypen werden also auf die Metadaten gemappt, um nur jene Dokumente zu durchsuchen, welche die Werte auch enthalten können (Abbildung 5). Dadurch kann der Document Crawler die Metadatensuche als Vorselektion der Dokumente verwenden, um nicht die kompletten Dokumente in allen Repositorys durchsuchen zu müssen.

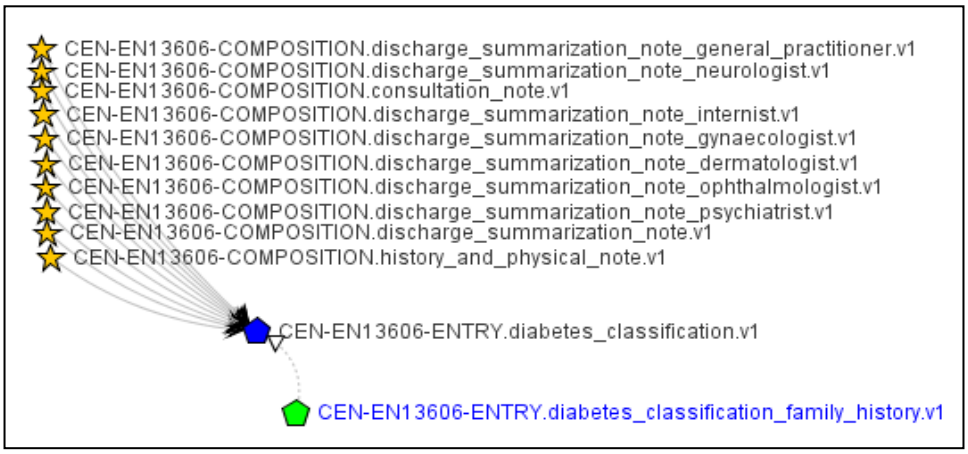

Abbildung 5: Infoitem 6.3 (Typ 1A Diabetes) kommt im Archetyp „diabetes_classification“ vor, also müssen nur die Dokumenttypen durchsucht werden, in denen dieser Archetyp vorkommt → entspricht allen COMPOSITION Archetypen die in der Grafik angezeigt werden.

Nach der Vorselektion wird auf das resultierende Dokumentenset die Archetypensuche angewandt. Dafür wird jedes Dokument einzeln aus dem jeweiligen Repository geladen und vom Document Crawler inhaltlich durchsucht. Das relevante Infoitem, welches das Referenzmodell über den Archetypen beschreibt, wird innerhalb des strukturierten Dokuments lokalisiert. Die gefundenen Ergebnisse werden anschließend zusammengefasst und an den Document Consumer retourniert.

Die Arbeitsschritte des Document Crawlers umfassen somit:

1. Suchanfrage des Document Consumers empfangen
2. Relevante Archetypen vom ATR holen
3. Relevante Dokumente mit Metadatensuche in Document Registry suchen
4. Gefundene Dokumente vom Document Repository laden
5. archetypenbasierte Suche auf Dokumente ausführen
6. Resultate zusammenfassen und an den Document Consumer zurückliefern

2.7.2.3 Archetype Repository

Das Archetype Repository, kurz ATR, beinhaltet alle Archetypen, welche für die Darstellung der medizinischen Sachverhalte im Bezug auf Diabetes Mellitus benötigt werden. Die Archetypen entsprechen dabei dem EN/SIO 13606 Standard. Jeder Archetyp enthält im optionalen Abschnitt namens „Ontology Section“ die zugehörigen Infoitems. Die Infoitems können außerdem gesammelt aus dem ATR extrahiert und im XML-Format zusammengefasst zur Verfügung gestellt werden (Abschnitt 3.3.2.1).

3. Methoden und Vorgehensplanung

Das ATR und der Document Crawler wurden vom Projektteam bereits zu einem großen Teil entwickelt. Die fehlende Komponente des Document Consumer muss noch im Konzept verfeinert und implementiert werden. Dieses Kapitel behandelt den Entwicklungsprozess der Schnittstellen im Document Consumer zum Anbinden an die erweiterten Akteure.

3.1 Vorgehensweise des Entwicklungsprozesses

Die Entwicklung der Schnittstellen beruht auf dem Erstellungsprozess des klassischen Software-Engineerings nach Balzert [29]. Dafür wurde ein erweitertes Wasserfallmodell verwendet [30]. Dieses erlaubt auch ein Zurückspringen zu einem vorherigen Entwicklungsschritt. Das Wasserfallmodell wurde auf die Problemstellung dieser Arbeit angepasst und durchläuft folgende Schritte:

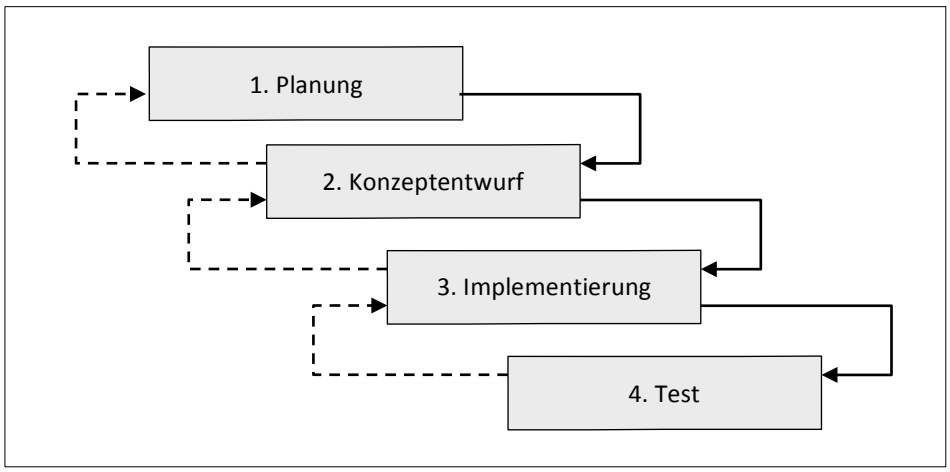

Abbildung 6: Erweitertes Wasserfallmodell angepasst an die Problemstellung dieser Arbeit

Die Planung (Abschnitt 3.2) beschreibt die Informationsbeschaffung zum Thema XML-Verarbeitung mit JAVA bzw. die Auswahl des geeignetsten XML-Parser-Frameworks für die Implementierung. Das Konzept der Schnittstelle im Document Consumer wird in Schritt 2 des Entwicklungsprozesses erstellt (Abschnitt 3.3). Der Konzeptentwurf wird somit vom Teilziel 1 dieser Arbeit abgedeckt. Die Implementierung (Abschnitt 3.4) beschreibt die Umsetzung des vorher

erarbeiteten Konzeptes als Ergebnis zu Teilziel 2. Anschließend folgt noch der Test der implementierten Schnittstellen (Abschnitt 3.5), welcher das Teilziel 3 dieser Arbeit beschreibt.

Allgemein kann anhand der Architektur des erweiterten XDS-Profils beim Document Consumer zwischen zwei Schnittstellen unterschieden werden:

1. Schnittstellen zwischen ATR und Document Consumer (Input)
2. Schnittstelle zwischen Document Crawler und Document Consumer (Output)

Der erwähnte Entwicklungsprozess wird somit ab dem Konzeptentwurf jeweils für beide Schnittstellen separat durchgeführt.

3.2 Planung

In diesem Abschnitt werden die Grundlagen für die Verarbeitung von XML-Dokumenten erarbeitet und eine geeignete Verarbeitungstechnik für die Problemstellung dieser Arbeit ausgewählt. Außerdem werden die grundlegenden Entwicklungstools festgelegt.

3.2.1 Auswahl des XML-Parser-Frameworks

Um XML-Dokumente in einem Programm nutzen zu können, bedient man sich einer geeigneten API[6] zur XML-Verarbeitung. Die Umwandlung einer XML-Datei in ein für die Programmiersprache geeignetes Format bzw. das schrittweise Einlesen der Datei wird auch Parsen genannt [31]. Der umgekehrte Weg, ein XML-Dokument aus einer spezifischen Datenrepräsentation eines Programms zu erzeugen, wird auch Serialisieren genannt [31]. Oft wird der Begriff „Parsen“ für die komplette XML-Verarbeitung, also Einlesen und Schreiben, verwendet.

Für die Problemstellung dieser Arbeit müssen zuerst verfügbare Frameworks ermittelt werden und anschließend das geeignetste Framework ausgewählt werden.

3.2.1.1 Erhebung verfügbarer XML- Parserframeworks durch qualitative Literaturrecherche

Anhand der Literaturrecherche sollen für JAVA[7] verfügbaren Frameworks zur Verarbeitung von XML-Dokumenten gefunden werden. Einerseits wurde dafür unter Wikipedia, Google bzw. Google-Scholar gesucht, um sich einen Überblick der verfügbaren Frameworks zu verschaffen. Die genaue Framework Beschreibung wurde dann von der jeweiligen Hersteller-Webseite bzw. aus einschlägiger Literatur bezogen.

[6] Application Program Interface – Anwendungsprogramm Schnittstelle
[7] ausschließlich JAVA-Frameworks, da die Implementierung in JAVA erfolgt

3.2.1.2 Ergebnisse der Literaturrecherche

Für XML-basierte Daten gibt es vier verschiedene Parsertypen in JAVA [31]:

- DOM-orientierte APIs
- Push-APIs
- Pull-APIs
- Mapping-APIs

Sie unterscheiden sich in der Art des Parsens bzw. in der JAVA-internen Speicherung der XML-Struktur. Die verschiedenen Parsertypen werden in den folgenden Abschnitten erläutert.

DOM

Das „Document Object Model", kurz DOM, ist ein vom W3C definierter Standard, der eine plattform- bzw. sprachenunabhängige Schnittstelle für den Zugriff auf XML definiert [32]. DOM repräsentiert dabei ein XML-Dokument im Programm als Baumstruktur. D.h. beim Einlesen eines Dokuments wird der gesamte Baum in den Speicher geladen. Der Vorteil von DOM besteht darin, dass die Struktur, nachdem sie einmal eingelesen wurde, vollständig zur Verfügung steht und somit leicht und schnell auf ein beliebiges Element im Baum zugegriffen werden kann. Außerdem ist DOM in der Lage XML-Dokumente zu erzeugen. DOM-orientierte APIs eignen sich nicht zum Verarbeiten sehr großer XML-Dokumente, da das komplette Einlesen sehr zeitaufwändig ist und die große Baumstruktur viel Speicher benötigt. DOM wird seit 1998 laufend weiterentwickelt und ist in verschiedenen Versionen, auch Levels genannt, verfügbar. Level 3 ist seit 2004 die aktuellste Version. Da die W3C-DOM Umsetzung programmiersprachenunabhängig ist, gibt es für JAVA speziell entwickelte DOM-Varianten. Die drei bekanntesten sind Dom4J [33], XOM [34] und JDOM [35].

JDOM ist eine open-soruce API und wurde im Jahr 2000 von Brett McLaughlin und Jason Hunter entwickelt. Ziel von JDOM ist es, alle Vorteile der DOM und SAX in einer API zu integrieren und speziell für JAVA zur Verfügung zu stellen. Ebenso wie DOM lädt JDOM die gesamte Baumstruktur in den Speicher; der Unterschied liegt allerdings darin, dass JDOM jeden Knoten als konkrete Klasse repräsentiert, während in DOM eine generische programmiersprachenunabhängige Klasse für alle unterschiedlichen Knoten verwendet wird. Die API ist aktuell in der Version 1.1.2 verfügbar.

Push-API

Eine Push-API verarbeitet einen XML-Stream sequentiell, d.h. das XML-Dokument wird im Unterschied zu DOM-APIs Schritt für Schritt geparst. Das funktioniert nach dem Call-Back-Prinzip:

Die XML Datei wird wie ein Datenstrom gelesen und bei bestimmten Inhalte wird ereignisgesteuert eine vorher definierte Call-Back Methode aufgerufen [31]. Stößt der Parser beim Einlesen auf ein XML-Element, wird die vom Benutzer definierte Methode „startElement()“ aufgerufen. Erkennt der Parser Text innerhalb des Elements, wird die Methode „characters()“ aufgerufen, welche den Text einliest. Wird das Element-Tag dann geschlossen folgt eine „endElement()“ Methode. So können die Elemente eines XML-Dokuments sequentiell eingelesen werden. Der Vorteil einer Push-API ist somit die speicherarme Abarbeitung beim Einlesen, womit auch die Verarbeitung von sehr großen Dokumenten ermöglicht wird. Der iterative Vorgang ist aber mit dem Nachteil verbunden, dass sich der Benutzer um die Weiterverarbeitung der gewonnen Daten kümmern muss und so ein Zugriff auf ein bestimmtes Element nicht ohne benutzerdefinierte Zwischenspeicherung möglich ist. Außerdem ist es mit einer Push-API nur möglich, XML-Dokumente zu lesen, jedoch nicht zu schreiben.

„Simple API for XML Parsing“, kurz SAX, ist der bekannteste JAVA Vertreter der Push-APIs [36]. Die API bietet Interfaces für JAVA an, um ein XML sequentiell nach dem Push-Ansatz zu parsen. SAX ist seit 2002 in der Version 2 verfügbar und gilt als De-facto-Standard, da es zwar von keinem offiziellen Gremium verabschiedet wurde, jedoch schnell Beliebtheit in der XML-Verarbeitung errungen hatte.

Pull-API

Eine Pull-API verarbeitet ein XML–Dokument ähnlich sequentiell wie eine Push-API. Der Unterschied ist, dass bei einer Push-API der Parser das Anwendungsprogramm über die Call-Back-Methoden aufruft, während bei einer Pull-API das Anwendungsprogramm iterativ den Parser aufruft. Somit wird bei einer Pull-API aktiv der nächste Teil des Dokuments angefordert. Dieses Prinzip entspricht dem Iterator Design Pattern [37]. Der Iterator muss somit vom Anwendungsprogramm von Element zu Element weiter bewegt werden. Es gibt zwei Verarbeitungstypen einer Pull-API:

1. Iterator-Verarbeitung
2. Curser-Verarbeitung

Bei der Iterator-Verarbeitung wird bei jedem neuen Element im XML ein neues Objekt erzeugt. Dieses Objekt kann in einer Datenstruktur gespeichert werden, um später darauf zurückgreifen zu können. Die Iterator-Verarbeitung ist somit flexibler als die Curser-Verarbeitung.

Bei der Curser-Verarbeitung wird das XML immer im selben Objekt weitergeparst. Somit steht zu einem bestimmten Zeitpunkt immer nur das Objekt mit der aktuellen Position zur Verfügung.

Die Curser-Verarbeitung ist zwar unflexibler, aber auch speichereffizienter und besitzt eine höhere Verarbeitungsgeschwindigkeit.

Der Vorteil der Pull-API ist die Kombination aus DOM und SAX: Es können auch große Dokumente geparst werden, die Kontrolle geht aber vom Anwendungsprogramm aus. Außerdem können mit Pull-APIs XML Dokumente sowohl gelesen als auch erzeugt werden.

„The Streaming API for XML" [38], kurz StAX, ist der Hauptvertreter der Pull-APIs.

Mapping-API

Mapping-APIs, bzw. auch XML-Binding genannt, versuchen anders als bei den vorher vorgestellten Parsertypen das Dokument direkt auf eine Klassenstruktur abzubilden. Das bedeutet, dass ein XML-Dokument ohne direkte Schnittstelle direkt in eine Klasseninstanz übergeführt wird. Vorteil der Mapping-APIs ist die leichte Verwendung in einer Applikation, da man sich den umfangreichen Parsvorgang spart. Das führt auch zu einem geringen Wartungsaufwand und einer geringen Fehleranfälligkeit. Die Nachteile ergeben sich aus den meist sehr umfangreichen Vorarbeiten, bis man den korrekten Mapping-Algorithmus generiert hat. Die bekannteste XML-Mapping-API für JAVA ist JAXB („JAVA API for XML Bindung") [39].

3.2.1.3 Auswahl des geeignetsten Parser-Frameworks anhand eines qualitativen Vergleichs

In diesem Abschnitt werden o.g. Parser-Frameworks untereinander verglichen. Dafür werden verschiedene Merkmale gegenüberstellt und bewertet. Die Merkmale umfassen dabei:

- Performance: wie effizient kann der Parser ein XML-Dokument einlesen
- Speicherintensität: wie viel Speicher benötigt der Parser für das Einlesen eines XML-Dokuments
- Erstellen von Dokumenten: kann mit dem Parser auch ein XML-Dokument erzeugt werden
- Zugriff auf bestimmten Knoten: wie flexibel/schnell kann auf einem bestimmten Knoten in der Struktur zugegriffen werden
- Programmieraufwand: welcher Aufwand muss für die Implementierung bzw. Anpassung der Parser-Schnittstellen aufgebracht werden
- Aufwand Objektabbildung: wie viel Code muss programmiert werden, um die geparsten Daten in ein Objekt zu bekommen
- Einarbeitungszeit: wie lange wird für das Erlernen der Parser-Syntax benötigt
- Dokumentation: wie gut ist verfügbare Dokumentation

Alle Merkmale werden für jedes Parser-Framework folgendermaßen bewertet:

Tabelle 4: Bewertungstabelle für die Merkmale der Parser-Frameworks

Bewertung	Bedeutung
++	Merkmal sehr gut bewertet bzw. Merkmal wird hervorragend unterstützt
+	Merkmal gut bewertet bzw. Merkmal wird unterstützt
-	Merkmal weniger gut bewertet bzw. Merkmal wird nur bedingt unterstützt
--	Merkmal sehr schlecht bewertet bzw. Merkmal wird nicht unterstützt

Tabelle 5: Vergleich der verschiedenen XML-Parsertypen

	Baum-orientiert (DOM)	Push (SAX)	Pull (StAX)	Mapping-API (JAXB)
Performance	-	++	+	+
Speicherintensität	--	++	++	+
Erstellen von Dokumenten	+	--	+	++
Zugriff auf bestimmtes XML-Element	++	-	+	+
Programmieraufwand Schnittstellen	++	--	-	--
Aufwand Objektabbildung	+	-	-	++
Einarbeitungszeit	+	+	-	--
Dokumentation	++	++	+	+

Für die Implementierung der Schnittstelle ist es von Bedeutung, das XML-Dokument so einzulesen, dass es im Programm leicht und effizient weiterverarbeitet werden kann. Der Zugriff auf ein bestimmtes Element sollte daher ohne viel Programmieraufwand möglich sein. Außerdem muss die Schnittstelle in der Lage sein ein XML-Dokument zu erstellen. Wie in der Gegenüberstellung aus Tabelle 4 zu erkennen ist eignet sich für die o.g. Anforderungen eine baum-orientierte API am besten.

Die Auswahl fiel letztendlich auf JDOM, da es eine DOM-API speziell für JAVA ist und durch die weite Verbreitung sehr gute Dokumentation zur Verfügung steht. Darüber hinaus sind viele Programmierbeispiele im Internet und einschlägiger Literatur zu finden, wodurch der Erlernungsprozess zusätzlich beschleunigt wird. JDOM wird auch von den anderen Erweiterungen des EHR-Arche Projekts verwendet, was ein Hinzufügen weiterer Bibliotheken ins Projekt erspart.

3.2.2 Verwendete Entwicklungstools

Alle Erweiterungen des Projekts EHR-Arche werden als Webservice mit dem ZKoss-Framework [40] in JAVA entwickelt. Die Schnittstellen werden somit auch in JAVA programmiert. Als Entwicklungsumgebung wird Eclipse in der Version 3.6 (Helios) verwendet [41]. Für das gemeinsame Arbeiten an der Projektimplementierung dient das Eclipse-Plugin „Subclipse“ [42].

Für die Entwicklung und Bearbeitung der XML-Dateien kommt ebenso Eclipse zum Einsatz, da die Entwicklungsumgebung einen guten XML-Editor beinhaltet. Zum Verarbeiten der XML-Dateien werden die JDOM Bibliotheken in der Version 1.1.2 integriert.

3.3 Zu Teilziel 1: Entwurf des Schnittstellenkonzeptes

In diesem Abschnitt wird das Konzept für die Schnittstelle des Document Consumers zum Document Crawler und ATR beschrieben. Als Vorgabe dienen dabei die vom Projektteam entwickelten Schnittstellenspezifikationen des ATR und des Document Crawlers. Da die Schnittstellen des Document Consumers auf die Funktionsweise der Suchanfragenerstellung in der erweiterten archetypenbasierten Suche aufbauen und es dafür noch kein konkretes Konzept gibt, muss vorher ein Entwurf zur Erstellung einer Suchanfrage anhand der vorhandenen Konzepte erarbeitet werden. Dieses Konzept zur Suchanfragenerstellung wurde zusammen mit Dr. Dipl.-Inf. Samrend Saboor und Dominik Schweiger erarbeitet.

3.3.1 Konzept für die Erweiterung des Document Consumers

Das Projektteam entwickelte bereits ein grobes Konzept für die Funktionalität der Erweiterung des Document Consumers. Dieses Konzept muss noch verfeinert werden, um daraus entsprechende Schnittstellen erstellen zu können.

Das bisherige Konzept schreibt vor, dass der Benutzer in der erweiterten Suche zwischen verschiedenen Sucharten wählen kann. Folgendes Anwendungsfalldiagramm verdeutlicht die unterschiedlichen Sucharten:

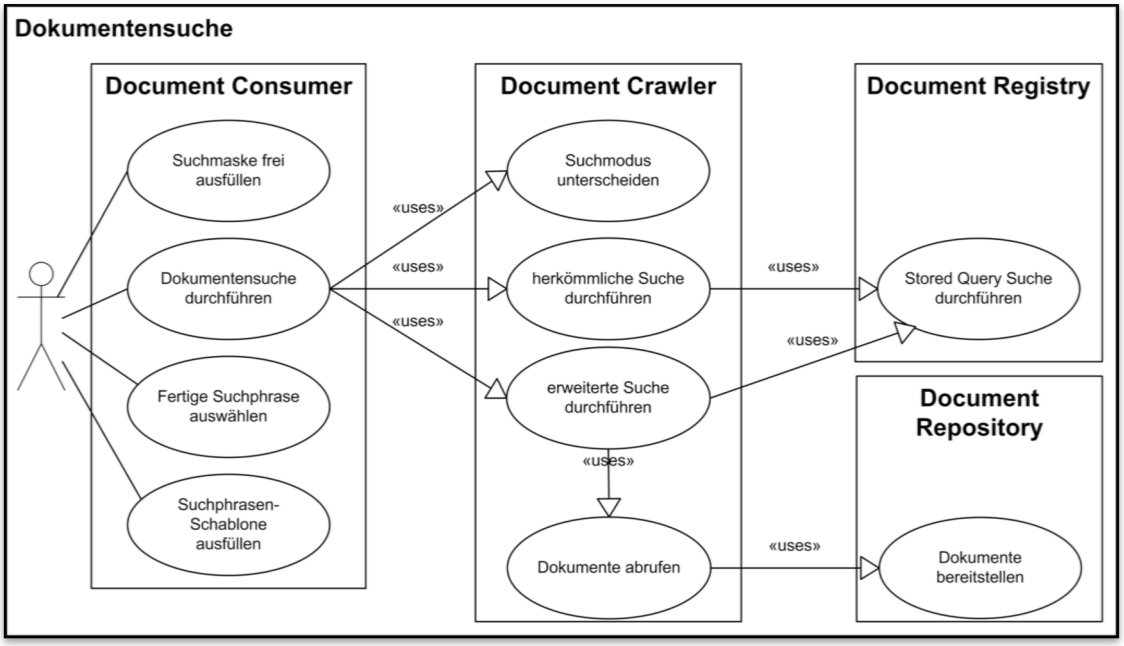

Abbildung 7: Vom Projektteam erstelltes Anwendungsfalldiagramm mit Funktionen der erweiterten XDS-Akteure

Es kann im Document Consumer zwischen drei Sucharten unterschieden werden:

1. Fertige Suchphrase auswählen: Fertige Suchphrasen beinhalten eine vordefinierte Kombination aus Infoitems und ihrer jeweiligen Werteeinschränkung, nach denen schnell, ohne weitere Benutzereingaben, gesucht werden kann. Eine fertige Suchanfrage könnte zum Beispiel das Erstgespräch sein, das immer die selben Infoitems für die Suche benötigt.
2. Suchphrasenschablone auswählen: Suchschablonen verwenden vordefinierte Infoitems, welche jedoch noch benutzerdefiniert parametrisiert werden können.
3. Suchmaske frei ausfüllen: Unter einer freien Suchmaske versteht man eine komplett selbst gebaute Suchanfrage, in welcher der Benutzer sowohl die Infoitems als auch die Werteeinschränkungen der Infoitems selbst auswählt und verknüpft. Die freie bzw. flexible Suche bildet die Basis der gesamten erweiterten Suche, da alle anderen Sucharten auf dieser Suche aufbauen können.

   Diese Art der Suche wurde jedoch zwecks Benutzerfreundlichkeit im finalen Prototypen des Document Consumers stark eingeschränkt. In der finalen Version kann der Benutzer nur ein Infoitem auswählen und dieses parametrisieren.

Bei allen drei Sucharten muss ein Zeitbereich festgelegt werden können, um die Suche einzugrenzen zu können (z.B. Dokumente der letzten 6 Monate). Außerdem setzen alle Sucharten voraus, dass die Infoitems im Document Consumer zur Verfügung stehen.

3.3.2 Kommunikation der erweiterten XDS-Akteure

Da XML flexibel einsetzbar und maschinenlesbar ist, entschied man sich im Projekt proprietäre XML-Nachrichten für die Kommunikation zwischen den erweiterten XDS-Akteuren zu verwenden. Somit müssen die Schnittstellen zum Document Consumer konforme XML-Nachrichten senden und empfangen können. Die Verbindung per XML gewährleistet technisch eine eindeutige Kommunikation zwischen Document Consumer und den neuen XDS-Komponenten. Die Inhalte der Nachrichten unterscheiden sich beim ATR- und Document Crawler-Akteur.

Grundlage für die Kommunikation mit dem ATR bilden die sogenannten Infoitems (Abschnitt 2.7.1). Jedes Infoitem repräsentiert dabei ein medizinisches Konzept (z.B. systolischer Blutdruck). Da die Infoitems bei allen drei Sucharten verwendet werden, müssen diese im Document Consumer zur Verfügung stehen. Der Document Consumer sendet dann eine Suchanfrage, welche die IDs der gewünschten Infoitems mit eventuell definierten Werteeinschränkungen beinhaltet, an den Document Crawler. Dieser kann aufgrund eines Infoitems auf den jeweiligen Archetyp schließen und somit die relevanten Dokumente finden (Abschnitt 2.7.2.2).

Allgemein muss der Document Consumer für die Kommunikation mit dem ATR und dem Document Crawler folgende Funktionen beherrschen:

- Den Empfang und die Verarbeitung des XML-Dokuments mit allen verfügbaren Infoitems aus dem ATR
- Die Erstellung einer XML-Query[8] nach der Schnittstellenspezifikation erläuterten Form mit Senden zum Document Crawler
- Den Empfang und die Verarbeitung des vom Document Crawler retournierten Ergebnisses

Die genauen Spezifikationen der Kommunikation werden in den nächsten Abschnitten erläutert.

3.3.2.1 Schnittstellenspezifikation ATR

Im ATR können alle Infoitems aus den Archetypen extrahiert werden. Die Liste der gesamten verfügbaren Infoitems wird im ATR als XML-Dokument zur Verfügung gestellt. In dieser XML-Datei werden die Infoitems in „nodes“ gegliedert, d.h. jedes „node“-Element repräsentiert ein Infoitem. Die Attribute eines „node“-Elements haben folgende Bedeutung:

- localCode: definiert einen eindeutige Nummer zur Identifizierung und Verarbeitung des Infoitems. Diese Identifizierung dient zur Zuordnung des Infoitems im Archetyp. Im Pro-

[8] Suchanfrage. Im Rahmen dieser Arbeit wird als Query die XML-Suchanfrage bezeichnet, welche an den Document Crawler gesendet wird.

jekt wurde dafür eine eindeutige, selbst definierte Nummer verwendet. Es könnte aber genauso ein SNOMED Code, ICD-10 o.ä. verwendet werden.

- labelDe,labelEn: beinhaltet die Beschreibung des Infoitems in englisch und deutsch

Die Infoitems sind hierarchisch gegliedert und bilden zwei Kategorien:

- **Primitive Infoitems:** elementare medizinische Werte, wie z.B. systolischer & diastolischer Blutdruckwert. Primitive Infoitems können bei einer inhaltlichen Suche durch Werteeinschränkungen spezialisiert werden (nur bestimmte Blutdruckwerte, etc.)
- **Komplexe Infoitems:** Container mit weiteren komplexen oder primitiven Infoitems (Blutdruckmessung, Allergien, etc.).

Komplexe Infoitems werden im Infoitem-XML[9] als „node"-Elemente angegeben, während primitive Infoitems mit einem „element"-Tag bezeichnet werden. Beide Arten besitzen die o.g. Attribute, jedoch beinhalten primitives Infoitems zusätzlich noch Elemente, welche für die Werteeinschränkung von Bedeutung sind. Die zusätzlichen Elemente primitiver Infoitems umfassen dabei[10]:

- Datatype: schreibt den Datentyp vor, welcher für die Parametrisierung eines Infoitems verwendet werden muss. Die Datentypen umfassen dabei[11]:
  - SIMPLE_TEXT: Freitext
  - PQ (Primitive Quality): primitiver Zahlentyp
  - IVL (Intervall): primitiver Zahlentyp im Intervall
  - DATE: Datum im Format *yyyy-MM-dd'T'hh:mm:ss*
  - BOOLEAN: Wahrheitswert true/flase
- Values: beinhaltet vordefinierte Werte in Englisch und Deutsch, welche zur Parametrisierung des Infoitems verwendet werden können
- Unit: gibt die Maßeinheit des Infoitems an, wenn dieses als Datentyp „PQ" besitzt

Abbildung 8 zeigt einen Ausschnitt aus dem vom ATR generierten Infoitems im XML-Format. „Laborbefund" repräsentiert ein komplexes Infoitem und beinhaltet die Items „Spezimen" und „Laboruntersuchung Blutbild". „Ergebnis" stellt ein primitives Infoitem dar, welches zwei Datentypen enthält:

1. Datentyp eines einfachen Textes (SIMPLE_TEXT), welcher die Auswahlmöglichkeiten „normal", „negativ" und „positiv" bereitstellt und in einer späteren Suche akzeptiert.

[9] Die XML-Datei, welche alle Infoitems strukturiert aus dem ATR enthält
[10] Die erweiterten Angaben entsprechen der „Element"-Klasse des EN/ISO 13606 Informationsmodells
[11] nach dem Datentypen-Standard CEN TS 14796

2. Einen frei wählbaren Zahlenwert (PQ), welcher die Einheit „IU/ml“ besitzt.

Bei der erweiterten Suche des Document Consumers soll der Benutzer aus allen Infoitems beliebig auswählen können. Während komplexe Infoitems nur zur Suche hinzugenommen werden können, ist bei primitiven Infoitems zusätzlich eine benutzerdefinierte Werteeinschränkung möglich. In Abbildung 8 kann man sich den „SIMPLE_TEXT“-Datentyp des Infoitems „Ergebnis“ als eine Auswahlliste vorstellen, welche für die Suche genau ein Element der „Value“-Blöcke zur Auswahl erlaubt. Besitzt ein Infoitem mehrere Datentypen, kann zwischen diesen frei gewählt werden. Man kann also das „Ergebnis“ in der Abbildung als „normal“, „negativ“ oder „positiv“ erfragen oder man gibt den genauen Wert in „IU/ml“ an.

```
<node localCode='60' labelDe='Laborbefund' labelEn='Laboratory findings'
count='1'>
   <node localCode='97' labelDe='Spezimen' labelEn='Specimen' count='1'/>
   <node localCode='80' labelDe='Laboruntersuchung Blutbild'
   labelEn='Laboratory test blood count' count='1'>
      <node localCode='80.3' labelDe='Hb' labelEn='HGB Hemoglobin' count='1'>
          <element localCode='21' labelDe='Ergebnis' labelEn='Result'
           count='1'>
              <SIMPLE_TEXT>
                 <value>
                     <de>normal</de>
                     <en>normal</en>
                 </value>
                 <value>
                     <de>negativ</de>
                     <en>negative</en>
                 </value>
                 <value>
                     <de>positiv</de>
                     <en>positive</en>
                 </value>
              </SIMPLE_TEXT>
              <PQ>
                 <unit>IU/ml</unit>
              </PQ>
          </element>
          .
          .
          .
```

Abbildung 8: Ausschnitt aus dem Infoitem-XML des ATR mit komplexen (z.B. Laborbefund) und primitiven Infoitems (z.B. Ergebnis)

3.3.2.2 Schnittstellenspezifikation Document Crawler

Der Document Crawler empfängt eine Suchanfrage als XML-Nachricht. Der Inhalt dieser Nachricht muss dabei der vom Projektteam entwickelten Syntax entsprechen, damit der Document Crawler die Suche ausführen kann.

Eine XML-Suchanfrage gilt für den Document Crawler als valide, wenn sie folgenden Anforderungen entspricht:

- Die XML-Nachricht ist wohlgeformt

- Das Wurzelelement wird mit „SearchQuery“ bezeichnet
- Es existiert ein Element „MetaData“: Dieses Element beinhaltet die allgemein beschreibenden Metadaten zur Suchanfrage, wie Name des Benutzers, welcher die Suche ausführt, Patientenstammdaten, etc.
- Die XML-Suchanfrage enthält mindestens:
    - ein „Tuple“-Element mit gültigem ItemCode aus dem ATR: Ein „Tuple“-Element beschreibt eine abgeschlossene Sucheinheit, welche genau ein Infoitem beschreibt. Ein Tuple enthält dabei folgende XML-Elemente:
        - ItemCode: Angabe des Infoitems anhand des ItemCodes
        - Unit: beschreibt die Einheit des Wertes in der Parametrisierung
        - Operator: enthält den Operator für die Werteeinschränkung; erlaubt sind die vom Projektteam entwickelten relationalen Operatoren:
            - EQ: equals (=)
            - NE: not equals (!=)
            - GT: greater than (>)
            - LT: lower than (<)
            - LE: lower equals (<=)
            - GE: greater equals (>=)
        - Value: beinhaltet den Wert der Parametrisierung des Infoitems
    - eine „Tuplecollection“ mit gültigem Verknüpfungsoperator: Die „Tuplecollection“ dient als Container weiterer Elemente. In jeder angeführten „Tuplecollection“ befinden sich somit mindestens 2 weitere Elemente, bestehend aus „Tuple“ oder weiteren „Tuplecollections“. Sie besitzt außerdem einen Verknüpfungsoperator, welcher die Beziehung der beinhalteten Elemente bei der Suche beschreibt. Die gültigen Verknüpfungsoperatoren sind dabei:
        - AND: alle beinhalteten Elemente müssen in einem Dokument enthalten sein, damit es als relevant gilt.
        - OR: nur ein Element innerhalb der Tuplecollection muss im Dokument enthalten sein, damit es als relevant gilt
- Die XML-Nachricht berücksichtigt die in Tabelle 6 erläuterte Semantik: Eine Suchanfrage enthält üblicherweise mehrere Infoitems, nach denen gesucht werden soll. Je nach dem, wie die Infoitems in der Suchanfrage kombiniert und strukturiert werden, wertet der Document Crawler die Anfrage unterschiedlich aus und liefert unterschiedliche Ergebnisse. Die Semantik des XML-Outputs der Suchanfrage muss daher die Infoitems je nach Fragestellung gemäß Tabelle 6 kombinieren.

Tabelle 6: Fragestellungen mit der zugehörigen Query-Semantik einer Suchanfrage

Fragestellung	Query der Suchanfrage	Beispielsuchanfrage	Aus dem Beispiel resultierende Pseudo-XML-Query
Man möchte alle Dokumente zu einem bestimmten Infoitem suchen	Query muss ein *Tuple* mit jeweiligem *ItemCode* des Infoitems aufweisen.	„liefere mir alle Dokumente, welche irgendwelche Werte der „klinischen Chemie" beinhalten (ItemCode 95)	<SearchQuery> <MetaData/> <TUPLE> <ItemCode>95</ItemCode> </TUPLE> </SearchQuery>
Man möchte Dokumente zu einem Infoitem suchen, welches nur bestimmte Ausprägungen ausweißt (z.B. bestimmter Laborparameter) → Suche nach primitiven Infoitem	Query muss ein *Tuple* mit jeweiligem *ItemCode*, dem gewünschten *Wert*, dem *Operator* und der *Einheit* aufweisen	„liefere mir alle Dokumente, in welchen ein Körpergewicht (ItemCode 3.1) größer 100kg dokumentiert ist"	<SearchQuery> <MetaData/> <TUPLE> <ItemCode>3.1</ItemCode> <Value>100</Value> <Operator>GT</Operator> </TUPLE> </SearchQuery>
Man möchte Dokumente finden, welche mehrere komplexe bzw. primitive Infoitems beinhalten	Alle *Tuple* müssen in einer *Tuplecollection* verschachtelt werden. Die *Tuplecollection* muss einen Verknüpfungsoperator aufweisen, welche das Verhältnis zu den beinhalteten *Tuple* für die Suche angibt	„liefere mir alle Dokumente, in denen eine Laboruntersuchung bezüglich Eisenstatus (IC 87) oder Elektrolytstatus (IC 93) stattgefunden hat"	<SearchQuery> <MetaData/> <TUPLECOLLECTION operator="OR"> <TUPLE> <ItemCode>87</ItemCode> </TUPLE> <TUPLE> <ItemCode>93</ItemCode> </TUPLE> </TUPLECOLLECTION> </SearchQuery>
Man verwendet in der Suchanfrage Infoitems, welche im ATR öfter vorkommen, z.B. das Infoitem mit dem ItemCode 21 „Ergebnis"	Damit der Document Crawler den Kontext erkennt, muss das übergeordnete Infoitem (z.B. Parathormon) in einer *Tuplecollection* zusammen mit dem Infoitem (z.B. Ergebnis) angegeben werden. Die *Tuplecollection* wird mit einem logischen *UND* verknüpft. Das Infoitem kann auch ohne Kontext angegeben werden, es werden dann alle Dokumente geliefert, welche irgendein Ergebnis beinhalten.	„liefere mir alle Dokumente, in denen ein Ergebnis (IC 21) zum Parathormon (IC 90.2) dokumentiert wurde"	<SearchQuery> <MetaData/> <TUPLECOLLECTION operator="AND"> <TUPLE> <ItemCode>21</ItemCode> </TUPLE> <TUPLE> <ItemCode>90.2</ItemCode> </TUPLE> </TUPLECOLLECTION> </SearchQuery>

In der Query in Abbildung 9 wird nach Dokumenten gesucht, in denen der ItemCode 93.2 (Kalium) vorkommt. Der ItemCode und die Einheit (falls vorhanden) stammen aus dem Infoitem-XML aus dem ATR. Ein Tuple kann außerdem über Werteeinschränkungen verfügen, wie Abbildung 9 im Tuple mit ItemCode 999. Dort wird festgelegt, dass das Erstellungsdatum der Dokumente größer (GT) „`2011-03-07T03:18:52`“ sein muss.

Die Zeitbereichs-Tuplecollection in Abbildung 9 beinhaltet je ein Tuple für die untere bzw. obere Schranke des Zeitbereichs. Beide Tuple sind mit „UND“ verknüpft. Das bedeutet, ein Dokument gilt bei der Suche nur als relevant, wenn beide Tuple zutreffen (und somit das Erstellungsdatum im Zeitbereich liegt). Der Zeitbereich besitzt immer ItemCode 999 und der Wert wird im Zeitformat *yyyy-MM-dd'T'hh:mm:ss* angegeben.

```
<?xml version="1.0" encoding="UTF-8"?>
<SearchQuery>
	<MetaData>
		<author>99999</author>
		<patient>123410101999</patient>
	</MetaData>
	<TUPLECOLLECTION operator="AND">
		<TUPLECOLLECTION operator="AND">
			<!-- Zeitbereich -->
			<TUPLE>
				<ItemCode>999</ItemCode>
				<Unit />
				<Operator>GT</Operator>
				<Value>2011-03-07T03:18:52</Value>
			</TUPLE>
			<TUPLE>
				<ItemCode>999</ItemCode>
				<Unit />
				<Operator>LT</Operator>
				<Value>2011-09-07T03:18:52</Value>
			</TUPLE>
		</TUPLECOLLECTION>
		<TUPLE>
			<!--Kalium -->
			<ItemCode>93.2</ItemCode>
			<Unit />
			<Operator />
			<Value />
		</TUPLE>
	</TUPLECOLLECTION>
</SearchQuery>
```

Abbildung 9: Beispiel einer validen XML-Query. Es wird nach Dokumenten zwischen dem 07.03.2011 und dem 07.09.2011 gesucht, in denen ein nicht näher definierter „Kalium“-Messwert vorkommt

3.3.3 Konzepterstellung der Document Consumer Schnittstelle

3.3.3.1 Konzept für die Erstellung und Darstellung einer Suchanfrage

Um flexible Suchanfragen im Document Consumer erstellen zu können, muss man nach den genannten Anforderungen (Abschnitt 3.3.2) beliebige Infoitems verknüpfen und parametrisieren können.

Diese Anforderung lässt sich am besten mit einem Editor umsetzen, der nach einem Baukastenprinzip arbeitet. Das bedeutet, es gibt Bausteine, welche vom Benutzer beliebig ausgewählt, kombiniert und strukturiert werden können. Die Bausteine entsprechen dabei den Infoitems. Ein Baustein kann zusätzlich noch parametrisiert werden, wenn es sich bei der Auswahl um ein primitives Infoitem handelt. Mehrere Bausteine können über einen Operator verbunden werden bzw. in einen Container mit eigenem Verknüpfungsoperator verpackt werden. Jede Zusammenstellung realisiert automatisch eine valide Suchanfrage, da es durch das Baukastenprinzip im Editor keine Möglichkeit gibt eine nicht vollständige Suchanfrage zu erstellen. Die Suchanfrage kann dann zum Ausführen an den Document Crawler gesendet werden.

3.3.3.2 Schnittstelle zum ATR

Für die Erzeugung einer Suchanfrage benötigt man die Infoitems aus dem ATR. Wie in der Schnittstellenspezifikation des ATR (Abschnitt 3.3.2.1) erläutert, können diese aus dem ATR extrahiert werden. Für die flexible Verwendung der Infoitems im Suchanfragen-Editor (Abschnitt 3.3.3.1), werden die Infoitems als JAVA-Objekte in das Programm integriert. Da die Infoitems als XML-Struktur vorliegen, können diese anhand eines Parser-Frameworks in den Document Consumer eingelesen werden. Dort können sie dann intern für die weitere Suchanfragenerstellung verwendet werden. Die Objektstruktur beinhaltet somit die Bausteine (Infoitems) für den Suchanfragen-Editor.

3.3.3.3 Schnittstelle zum Document Crawler

Die Suchanfrage liegt nach der Erstellung im Document Consumer in JAVA-Objekten gespeichert vor. Damit eine Suchanfrage an den Document Crawler geschickt werden kann, muss diese zuerst im Document Consumer in eine valide, der Spezifikation (Abschnitt 3.3.2.2) entsprechenden, XML-Query serialisiert werden.

Man kann bei einer Suchanfrage zwischen zwei Teilen unterscheiden:

1. Das Infoitem, welches eventuell durch Werte eingeschränkt wird
2. Die Zusammenstellung der verschiedenen Infoitems für die Suche

Da eine Suchanfrage aus vielen Infoitems aufgebaut ist und diese bereits strukturiert im Document Consumer zur Verfügung stehen, kann auf die vorhandene Struktur bei der Erstellung der Suchanfrage zurückgegriffen werden. Es macht Sinn, die Benutzereingaben (Werteeinschränkung) direkt im jeweiligen Knoten der erzeugten Baumstruktur zu integrieren. Dies hat den Vorteil, dass alle relevanten Daten, wie dazugehöriger Datentyp, Eltern-Kind Konstellation, etc. zusammen in einem Objekt gespeichert sind. Aus der Struktur kann dann die valide XML-Suchanfrage erstellt werden. Diese Anfrage wird anschließend an den Document Crawler zur Auswertung gesendet.

Als Ergebnis der Suchanfrage, liefert der Document Crawler ein XHTML-File mit den gefundenen Werten und dem Link zum jeweiligen Dokument im Repository an den Document Consumer zurück. Für die Darstellung des XHTML ist im Document Consumer keine weitere Schnittstelle erforderlich, da dieses direkt im Front-End dargestellt werden kann.

3.4 Zu Teilziel 2: Implementierung

Dieser Abschnitt beschreibt die Methodik zur Implementierung der Schnittstellen nach dem vorher entwickelten Schnittstellenkonzept (Abschnitt 4.1).

3.4.1 Schnittstelle zwischen ATR und Document Consumer

Diese Schnittstelle bindet an das ATR und lädt die Infoitems in den Document Consumer, wo sie gespeichert werden und später für den Suchanfragen-Editor zur Verfügung stehen.

Da die Infoitems des ATR bereits hierarchisch strukturiert im XML-Format vorliegen, wird im Document Consumer ebenso eine hierarchische Struktur zur Speicherung verwendet. In JAVA bietet sich dafür am besten eine Baumstruktur an. Weil die Infoitem-Struktur aus dem ATR zu komplex für eine Abbildung auf eine Standardbaumstruktur wäre, wurde eine proprietäre Node-Datenstruktur erstellt. Jeder Knoten in der Struktur besitzt zusätzlich zu den beschreibenden Informationen (*itemCode, label*) des Infoitems auch noch einen Zeiger auf den Elternknoten bzw. eine Liste bestehend aus Zeigern auf seine Kinder. Primitive und komplexe Infoitems werden in der selben Node-Struktur gespeichert, obwohl ein primitives Infoitem mehr Speicher für zusätzliche Informationen benötigt. Bei einem komplexen Infoitem bleiben diese erweiterten Strukturen leer. Um den Infoitem-Baum in JAVA zu erzeugen, wird ein neues Node-Objekt erstellt, welches den Wurzelknoten des XMLs aus dem ATR übergeben bekommt. Dann wird im Konstruktor das XML Schritt für Schritt mit JDOM[12] geparst und daraus rekursiv die Baumstruktur anhand

[12] in JDOM wird hier der SAX-Parser verwendet

der Node-Objekte erstellt. Das Parsen findet somit in jedem Node-Konstruktoraufruf statt. Aus dem XML Ausschnitt des ATRs in Abbildung 10 wird somit eine Datenstruktur in Abbildung 11 erstellt.

```
<node localCode='95' labelDe='Klinische Chemie' labelEn='Clinical chemistry'
count='1'>
    <node localCode='122' labelDe='Laboruntersuchung Vitamin B12'
    labelEn='Laboratory test vitamin b12' count='1'>
        <node localCode='122.2' labelDe='Vitamin B12' labelEn='Vitamin B12'
        count='1'/>
    </node>
    <node localCode='96' labelDe='Hormone' labelEn='Hormone' count='1'>
        <node localCode='81' labelDe='Laboruntersuchung Pankreas'
        labelEn='Laboratory test Pancreas' count='1'>
            <node localCode='81.2' labelDe='Insulin' labelEn='Insulin'
            count='1'/>
        </node>
        <node localCode='82' labelDe='Laboruntersuchung Nebenniere'
        labelEn='Laboratory test Adrenal gland' count='1'>
            <node localCode='82.2' labelDe='Cortisol' labelEn='Cortisol'
            count='1'/>
        </node>
    </node>
</node>
```

Abbildung 10: Ausschnitt aus dem Infoitem-XML

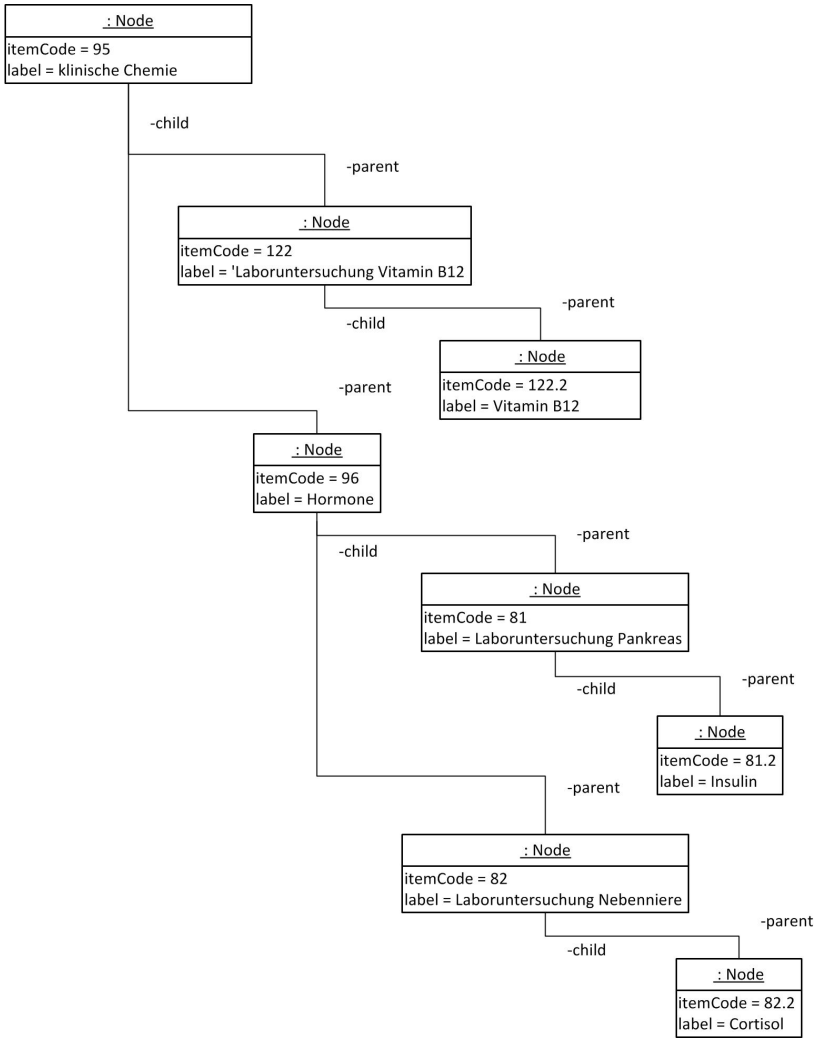

Abbildung 11: Generierte Baumstruktur nach dem Parsen des Infoitem-XML-Ausschnitts aus Abbildung 10

Stößt der Algorithmus beim sequentiellen Parsen im XML auf einen Datentyp, muss dieser zusammen mit allen weitern Inhalten wie *Values*, *Units*, etc. im Node gespeichert werden. Dazu dient die Klasse DataContainer. Ein DataContainer-Objekt beinhaltet die genauen Informationen für einen Datentyp zur Werteeinschränkung des Archetyps. Das DataContainer-Objekt einer Node-Instanz beinhaltet somit nach dem Parsen alle XML-Elemente aus dem Infoitem-XML, welche innerhalb eines primitiven Infoitems noch vorkommen. Ein Knoten im XML kann auch mehrere Datentypen enthalten, wodurch auch mehrere DataContainer möglich sein müssen. Ein Node-Objekt beinhaltet somit eine Liste, welche beim Parsen eines primitiven Infoitem dynamisch mit DataContainer-Objekten gefüllt wird.

3.4.2 Schnittstelle zwischen Document Crawler und Document Consumer

Wie im Konzept (siehe Abschnitt 3.3.3.3) erläutert, werden die benutzerdefinierten Daten (Werteeinschränkungen primitiver Infoitems) zur Erstellung einer Suchanfrage direkt im jeweiligen Knoten integriert, welcher bereits aus dem Infoitem-XML erzeugt wurde (Abschnitt 3.4.1).

Jeder Node-Knoten enthält somit zur Verwaltung der benutzerdefinierten Angaben folgende Datenelemente:

- Daten, welche den Input des Benutzers zum Infoitem enthalten:
  - Wert bzw. Werteintervall
  - Operator
- Daten, welche bereits in der Node-Struktur enthalten sind, da sie aus dem ATR übernommen wurden (Inhalt des DataContainers)
  - Datentyp
  - Wertevorgaben
  - Maßeinheit

Die Input-Daten des Benutzers können beim Erstellen einer Suchanfrage im Front-End des Document Consumer über den Suchanfragen-Editor (Abschnitt 3.3.3.1) eingegeben und bearbeitet werden. Dort ist auch eine Werteeinschränkung eines Infoitems möglich. Für die Benutzereingabe wird kein neues Objekt erstellt, der Benutzer bearbeitet direkt den jeweiligen Node-Knoten.

Für die Zusammenstellung verschiedener Infoitems in einer Suchanfrage reicht die Node-Struktur nicht mehr, da nicht alle Nodes (wie sie aus dem ATR kommen) für die Suche verwendet werden bzw. es theoretisch auch möglich wäre, selbe Nodes öfter in einer Suche zu verwenden. Daher wurde eine neue Struktur „SearchQuery“ implementiert, welche die für die Suche gewünschten Infoitems beliebig zusammenstellen kann. Die SearchQuery-Klasse enthält also Node-Objekte für die Suche.

Aus der SearchQuery wird dann die fertige XML-Suchanfrage erstellt. Eine SearchQuery Instanz enthält folgende Teile:

- Metadaten
- Zeitbereich für die Suche
- Menge an Infoitems (Node-Objekten) verknüpft mit einem Verbindungsoperator

Die Infoitems hätten auch in einer Liste aus Node-Objekten gespeichert werden können. Dies ist jedoch unflexibel, da es im Suchanfragen-Editor auch möglich sein soll, verschachtelte Strukturen anhand von Containern zu erstellen, welche wiederum Infoitems beinhalten (Abschnitt 4.1.1). Somit wurde für die gewählten Infoitems ebenso eine Baumstruktur zur Speicherung verwendet. Da die Infoitems bereits als Node-Objekte vorliegen, wurde hier die selbe Node-Struktur angewandt.

Die in Abschnitt 4.1.1 entwickelte Lösung zur Erstellung eines verschachtelten Containers aus Infoitems wird in der Implementierung durch einen Zeiger und Klammern geregelt. Der Zeiger befindet sich immer an der Position, an der das nächste Infoitem zur SearchQuery hinzugefügt wird. Man kann sich das wie den Lesemarker (Curser) in einem Textverarbeitungsprogramm, z.B. Microsoft Word, vorstellen: Immer dort wo sich der Cursor befindet, wird das nächste Zeichen, Bild, Diagramm, etc. eingesetzt. Wird zu Beginn (d.h. zur leeren SearchQuery) ein Infoitem zur Struktur hinzugefügt, befindet sich dieses Item in der 1. Ebene. Alle weiteren Infoitems werden in der gleichen Ebene hinzugefügt. Eine Ebene besitzt immer einen Verknüpfungsoperator, das heißt, alle Elemente in der Ebene werden mit dem selben Operator verknüpft.

Für das Wechseln zu einer neuen Ebene (und somit das Erstellen eines Containers) wird eine Klammerfunktion erstellt: Öffnet man in der SearchQuery eine Klammer, springt der Cursor eine Ebene tiefer. Wird die Klammer geschlossen, wird der Container verlassen und zur ursprünglichen Ebene zurückgekehrt. So kann Schritt für Schritt eine Struktur aus Infoitems und Container für die Suchanfrage aufgebaut werden.

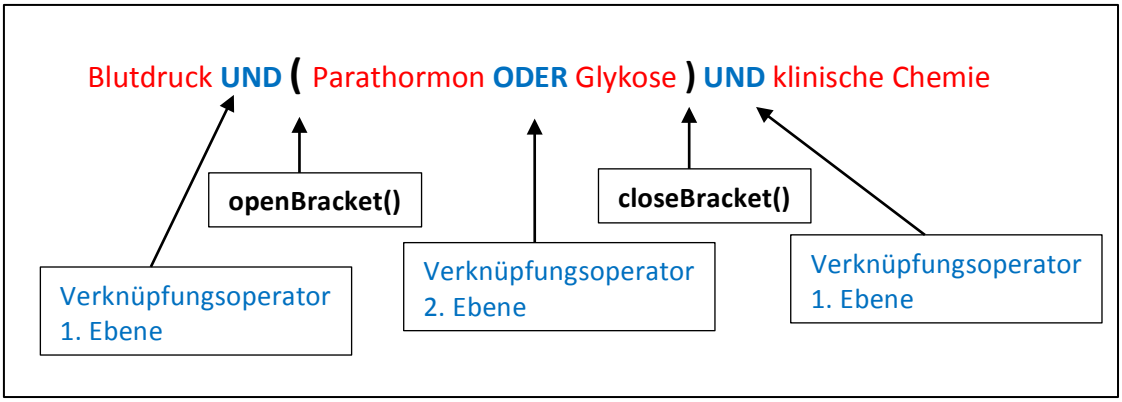

Abbildung 12: Beispiel einer Suchanfrage und wie diese theoretisch in der SearchQuery-Klasse aufgebaut wird. Jeder Ebene werden die Infoitems (rot) mit einem benutzerdefinierten Operator verknüpft. Mit der openBracket() Methode kann eine neue Ebene mit eigenem Verknüpfungsoperator erstellt werden.

Aus dieser SearchQuery-Struktur muss dann die XML-Query nach den in der Spezifikation (Abschnitt 3.3.2.2) erläuterten Richtlinien erzeugt werden. Dazu wird aus jedem Infoitem im Baum ein XML-Element mit der Bezeichnung „TUPLE“ und den beinhalteten Parametern aus dem Node-Objekt rekursiv erstellt. Alle Infoitems im Baum, welche sich in gleicher Ebene der Search-Query befinden, werden zu einem XML-Element „TUPLECOLLECTION“ mit entsprechendem Verknüpfungsoperator gruppiert. Metadaten und Zeitbereich der Suchanfrage werden ebenso als XML-Elemente in der Query definiert.

JDOM bietet dann eine Klasse Document an, aus welcher ein XML-Dokument serialisiert werden kann. Eine Document-Instanz beinhaltet dabei die gesamten Elemente, welche aus dem vorher genannten Prozess erstellt wurden. Die erstellte XML-Query kann dann vom Document Crawler ausgewertet werden.

3.5 Zu Teilziel 3: Test der Schnittstellen

Für jede Schnittstelle wurde ein Modultest durchgeführt, um die Funktionalität der Implementierung zu überprüfen. Ein Modul entspricht dabei genau einer Schnittstelle. Anschließend wurden die Schnittstellen im Document Consumer integriert und ein Integrationstest durchgeführt. Bei den Tests handelt es sich ausschließlich um Regressionstests [43], wobei das Soll-Ergebnis in den Schnittstellenspezifikationen (Abschnitt 3.3.2) beschrieben ist.

3.5.1 Test der ATR Schnittstelle

Zum Testen wurde ein einfaches Testprogramm entwickelt. Dieses Programm bekommt den Link des „infoitem.xml“ übergeben und speichert die komplette Sammlung an verfügbaren Infoitems in der Node-Datenstruktur. Anschließend werden nacheinander die Informationen aller Nodes in der Konsole ausgeben. Aus der Schnittstellenspezifikation des ATR aus Abschnitt 3.3.2.1 ist zu erkennen, dass beim Infoitem-XML folgende syntaktische Fälle auftreten, welche vom Pars-Algorithmus unterschiedlich zu behandeln sind:

- **Wurzelknoten**: schließt alle anderen Elemente im ATR-XML ein, besitzt localCode=’0’ und weist keinen Elternknoten auf
- **Einfacher Knoten**: stellt ein komplexes Infoitem dar, im Infoitem-XML zu erkennen am „<node></node>“-Tag
- **Knoten mit erweiterter Information**: stellt ein primitives Infoitem dar, zu erkennen am „<element></element>“-Tag bzw. an den darin enthaltenen Datentypen

Im letzten Punkt unterscheiden sich weitere Fälle:

- Element besitzt **einen** Datentyp: z.B. Infoitem „Wirkstoff Kategorie“ → nur SIMPLE_TEXT Datentyp
- Element besitzt **zwei** Datentypen: z.B. Infoitem „Ergebnis“ → SIMPLE_TEXT oder PQ Datentyp

Für jedes dieser Anwendungsfälle wurde aus dem Infoitem-XML ein Beispiel ausgewählt und dann überprüft, ob die entwickelte Baumstruktur die Information richtig speichert bzw. wiedergibt. Dafür wurde im Testprogramm eine einfache SWING-GUI implementiert und darin der Baum grafisch dargestellt. Durch Anklicken eines Knotens wird die im Knoten gespeicherte Information inklusive der Inhalte der DataContainer-Objekte in der Konsole ausgegeben. Der Test wurde dann manuell anhand eines Vergleichs der Ausgabe (Ist-Ergebnis) mit dem gewünschten Ergebnis aus der Spezifikation (Soll-Ergebnis, Abschnitt 3.3.2.1) durchgeführt.

3.5.2 Test der Document Crawler Schnittstelle

Für den Test der Document Crawler Schnittstelle wurde ein Programm erstellt, welches aus bestimmten Nodes (Infoitems) eine SearchQuery zusammenstellt und daraus die Suchanfrage nach den Spezifikationen aus Abschnitt 3.3.2.2 aufbaut. Die verwendeten Infoitems werden dabei direkt im Programmcode hinzugefügt und bearbeitet. Eine eigene Testklasse dient dazu, die Schritte aus dem Flussdiagramm in Abbildung 13 sequentiell in der Main-Methode abzuarbeiten.

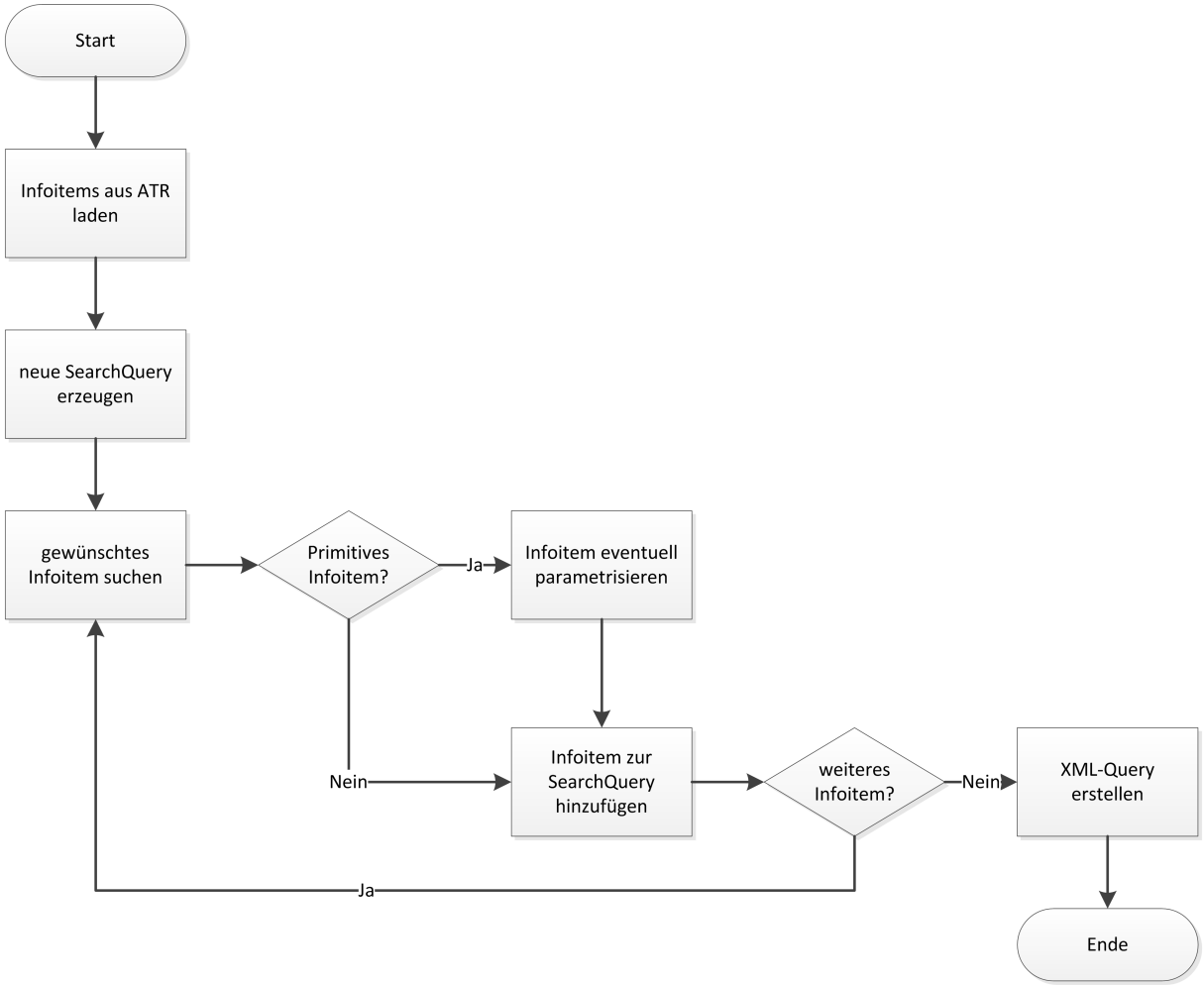

Abbildung 13: Ablauf des Programms zum Testen der Document Crawler Schnittstelle. So kann eine beliebige Suchanfrage mit beliebigen Infoitems erstellt werden.

Zum Testen wurden nach o.g. Schema einfache fiktive Querys erzeugt, deren Ausgaben (Ist-Ergebnis) mit der Spezifikation (Soll-Ergebnis, Abschnitt 3.3.2.2) verglichen wurden.

Die vom Document Crawler retournierte XHTML-Antwort wurde zum Test nur ausgegeben. Damit wurde überprüft, dass der Empfang der XHTML-Antwort funktionierte (Abschnitt 3.3.3.3).

3.5.3 Integrationstest im Document Consumer

Nachdem die Schnittstellen wie gewünscht funktioniert hatten, konnten sie in den Document Consumer integriert werden. Der Einbau der Schnittstellen in den Consumer wurde von Dominik Schweiger durchgeführt, da die Schnittstellen mit dem Editor im Front-End verbunden werden müssen. Der Integrationstest der Schnittstellen konnte dann im fertigen Document Consumer über das Ausführen von Beispielsuchen durchgeführt werden.

4. Ergebnisse

4.1 Zu Teilziel 1: Konzept für die Schnittstellen zu den erweiterten XDS-Akteuren

4.1.1 Konzept zur Erstellung einer Suchanfrage

Das Baukastenprinzip erlaubt es, aus den vorhandenen 446 Infoitems beliebige auszuwählen und daraus eine flexible Suchanfrage zu erstellen. Die Infoitems können im Suchanfragen-Editor vom Benutzer frei kombiniert, strukturiert und parametrisiert werden. Die einzelnen Funktionen des Editors wurden als Use Cases definiert, welche in Abbildung 14 dargestellt sind. Alle Use Cases werden in den folgenden Abschnitten genauer beschrieben.

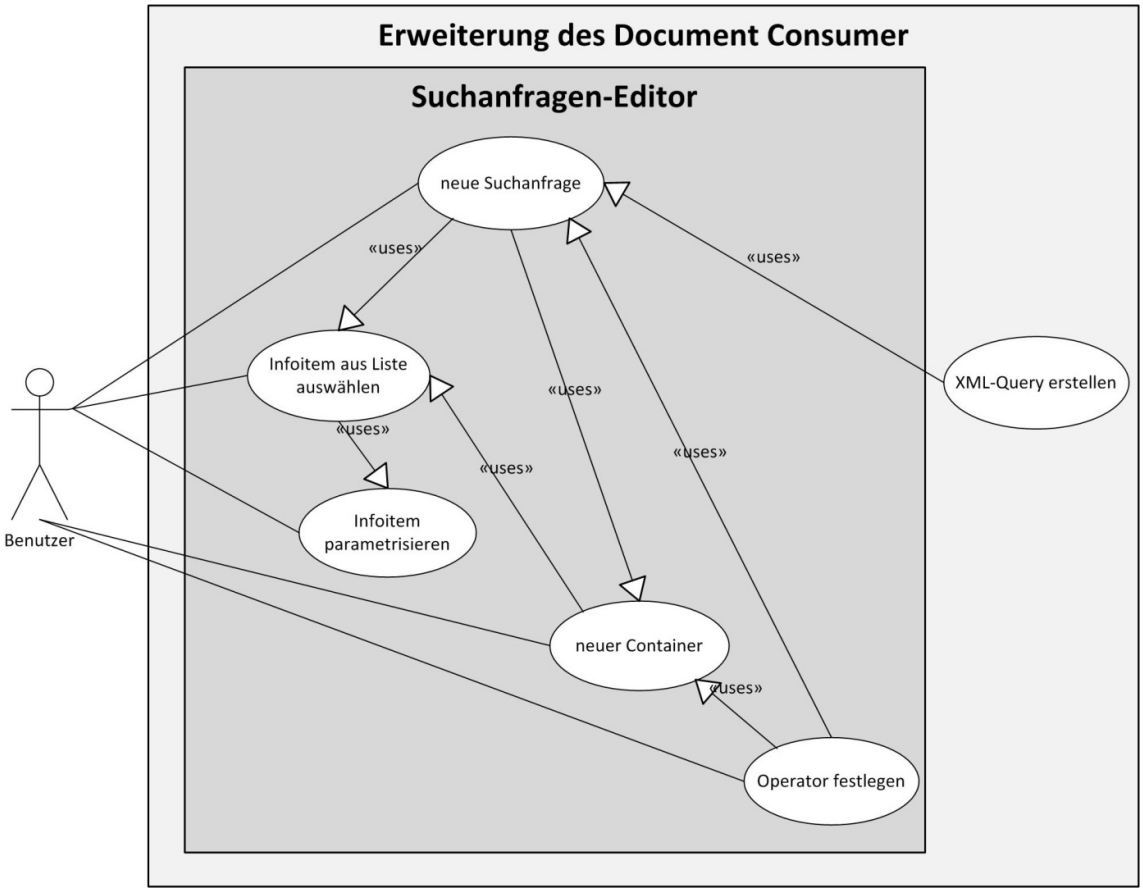

Abbildung 14: Anwendungsfalldiagramm, welches die Use Cases des Suchanfragen-Editors darstellt.

4.1.1.1 Use Case 1: neue Suchanfrage

Beschreibung	Der Benutzer erstellt eine neue (leere) Suchanfrage. Beim Start des Editors wird automatisch eine neue Suchanfrage angelegt. Diese entspricht einem Container mit eigenem Verknüpfungsoperator in oberster Ebene
Auslöser	• Start des Suchanfragen-Editors • Benutzer erstellt neue Suchanfrage
Vorbedingungen	

4.1.1.2 Use Case 2: Infoitem aus Liste auswählen

Beschreibung	Der Benutzer wählt aus der Liste aller verfügbaren Infoitems ein Infoitem aus.
Auslöser	Benutzer sucht ein neues Infoitem
Vorbedingungen	• Neue Suchanfrage muss erstellt sein (Use Case 1) • Neuer Container wurde erstellt (Use Case 4)

	• Alle verfügbaren Infoitems müssen aus dem ATR geladen sein

4.1.1.3 Use Case 3: Infoitem parametrisieren

Beschreibung	Der Benutzer wählt die gewünschte Werteeinschränkung und den entsprechenden relationalen Operator für das Infoitem aus.
Auslöser	Primitives Infoitem wurde ausgewählt
Vorbedingungen	• Entsprechendes Infoitem suchen (Use Case 2)

4.1.1.4 Use Case 4: Container erstellen

Beschreibung	Der Benutzer erstellt einen Container, welcher weitere Infoitems oder Container enthält. Alle darin befindlichen Elemente werden mit einem eigenen Operator verknüpft.
Auslöser	Benutzer erstellt neuen Container
Vorbedingungen	• Neue Suchanfrage muss erstellt sein (Use Case 1)

4.1.1.5 Use Case 5: Operator festlegen

Beschreibung	Der Benutzer wählt einen Operator für die Suchanfrage oder einen Container fest
Auslöser	Benutzer bestimmt Operator
Vorbedingungen	• Neue Suchanfrage wurde erstellt (Use Case 1) → Operator für Suchanfrage bestimmen • Neuer Container wurde erstellt (Use Case 4) → Operator für Container bestimmten

4.1.1.6 Use Case 6: XML-Query erstellen

Beschreibung	Die erstellte Suchanfrage wird in eine XML-Query transformiert
Auslöser	Benutzer führt die Suchanfrage aus
Vorbedingungen	• Neue Suchanfrage muss erstellt sein (Use Case 1) • Suchanfrage besteht aus beliebigen Infoitems und Containern → Benutzer hat gewünschte Suchanfrage erstellt

4.1.2 Schnittstellenkonzept des Document Consumers

Abbildung 15 beschreibt das fertige Konzept der Erweiterung des Document Consumers mit seinen Schnittstellen zu den erweiterten XDS-Akteuren. Der Document Consumer ist dabei in ein Back-End (Funktioneller Teil) und ein Front-End (grafische Darstellung und Benutzerschnittstelle) geteilt (Abschnitt 2.7.2.1). Wird der Suchanfragen-Editor im Front-End gestartet bzw. eine neue Suchanfrage erstellt, werden im Back-End automatisch die Infoitems aus dem ATR geladen. Dabei wird das Infoitem-XML, welche alle Infoitems im ATR strukturiert im XML-Format beinhaltet, im Document Consumer zu JAVA-Objekten geparst. Die Liste aller Infoitems steht nun dem Suchanfragen-Editor (Abschnitt 4.1.1) als JAVA-Objekte zu Verfügung, wo sie für die Erstellung einer Suchanfrage vom Benutzer verwendet werden. Nachdem die Suchanfrage im Editor fertig erstellt wurde, muss diese von einer JAVA-Struktur in eine valide XML-Query transformiert werden. Dies übernimmt der XML-Serialisierer im Back-End. Die erzeugte XML-Query wird dann an den Document Crawler zur Auswertung gesendet.

Die Ergebnisse werden anschließend vom Document Crawler im XHTML-Format retourniert. XHTML kann ohne weitere Verarbeitung direkt im Front-End dargestellt werden.

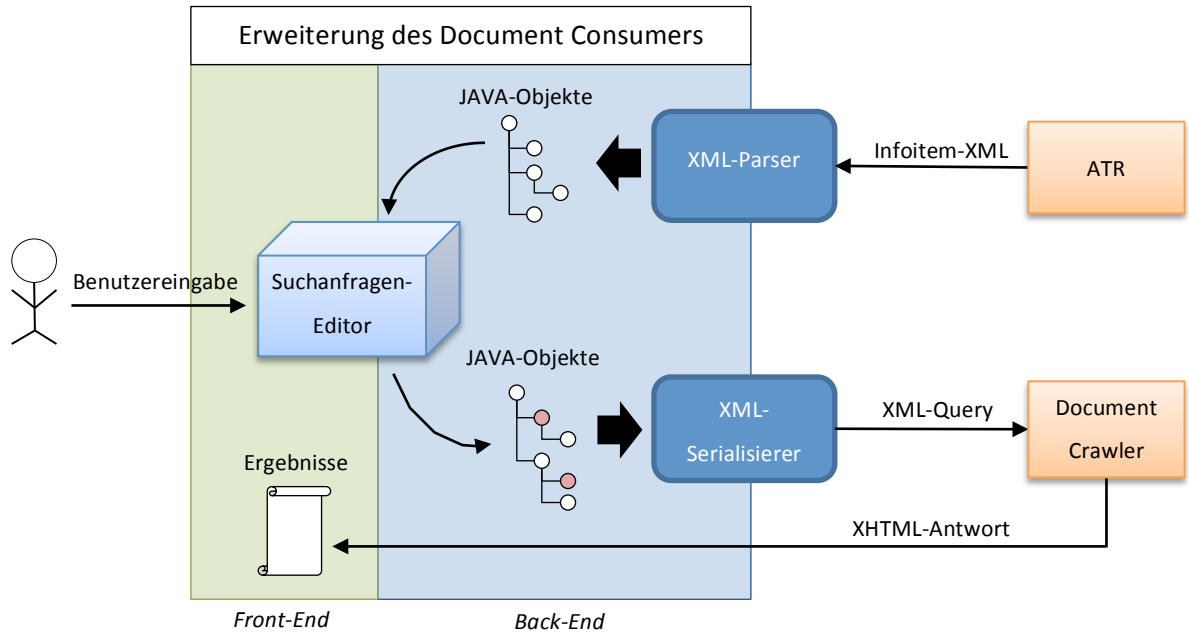

Abbildung 15: Aufbau der Erweiterung des Document Consumers für die Erstellung inhaltlicher Suchanfragen mit den Schnittstellen und deren Verbindung zum ATR und Document Crawler. Wird eine neue Suchanfrage im Suchanfragen-Editor erstellt, werden alle Infoitems aus dem ATR geladen und in eine JAVA-Struktur geparst. Nachdem der Benutzer die Suchanfrage erstellt hat, wird die JAVA-Struktur in eine XML-Query serialisiert und an den Document Crawler geschickt. Die Ergebnisse werden anschließend als XHTML retourniert und direkt im Front-End präsentiert.

4.2 Zu Teilziel 2: Implementierung der Schnittstellen

In diesem Abschnitt werden die Ergebnisse der Implementierung präsentiert. Die Implementierung setzt dabei das Schnittstellenkonzept (Abschnitt 4.1.2) um und dient als Grundgerüst für das Konzept des Suchanfragen-Editors (Abschnitt 4.1.1). Im Suchanfragen-Editor werden nur die Back-End Funktionen umgesetzt. Darunter zählt: die Infoitems aus dem ATR laden und eine Struktur zur Erstellung und Speicherung einer Suchanfrage zur Verfügung zu stellen, welche sowohl alle Funktionen des Editors unterstützt (Abschnitt 4.1.1), als auch einfach in eine valide XML-Query (Abschnitt 3.3.2.2) transformiert werden kann.

Die Implementierung der Schnittstellen besteht im Allgemeinen aus drei Klassen:

- Node-Klasse: dient zur Erstellung und internen Speicherung der Infoitem-Struktur mit allen Eigenschaften und zur Speicherung der vom Benutzer eingegebenen Werteeinschränkung.
- DataContainer-Klasse: enthält wichtige Informationen aus dem ATR, welche für die Parametrisierung eines primitiven Infoitems dienen.
- SearchQuery-Klasse: verwaltet die Erstellung einer Suchanfrage. Einzelne Nodes werden dabei hierarchisch verschachtelt. Aus dieser Hierarchie wird dann eine für den Document Consumer konforme XML-Anfrage generiert.

Die genaue Funktionsweise dieser Klassen wird in den nächsten Abschnitten beschrieben. Zusätzlich zu diesen Klassen existieren noch zwei Enumerations:

- RelationalOperator: beinhaltet alle verfügbaren Operatoren zur Werteeinschränkung eines primitiven Infoitems (Abschnitt 3.3.2.2)
- Datatype: beinhaltet alle verfügbaren Datentypen, die ein primitives Infoitem besitzen kann (Abschnitt 3.3.2.1)

In Abbildung 16 wird der Zusammenhang der Klassen und Enumerations dargestellt. Die rot markierten Teile sind dabei ausschließlich für die Parametrisierung eines primitiven Infoitems nötig.

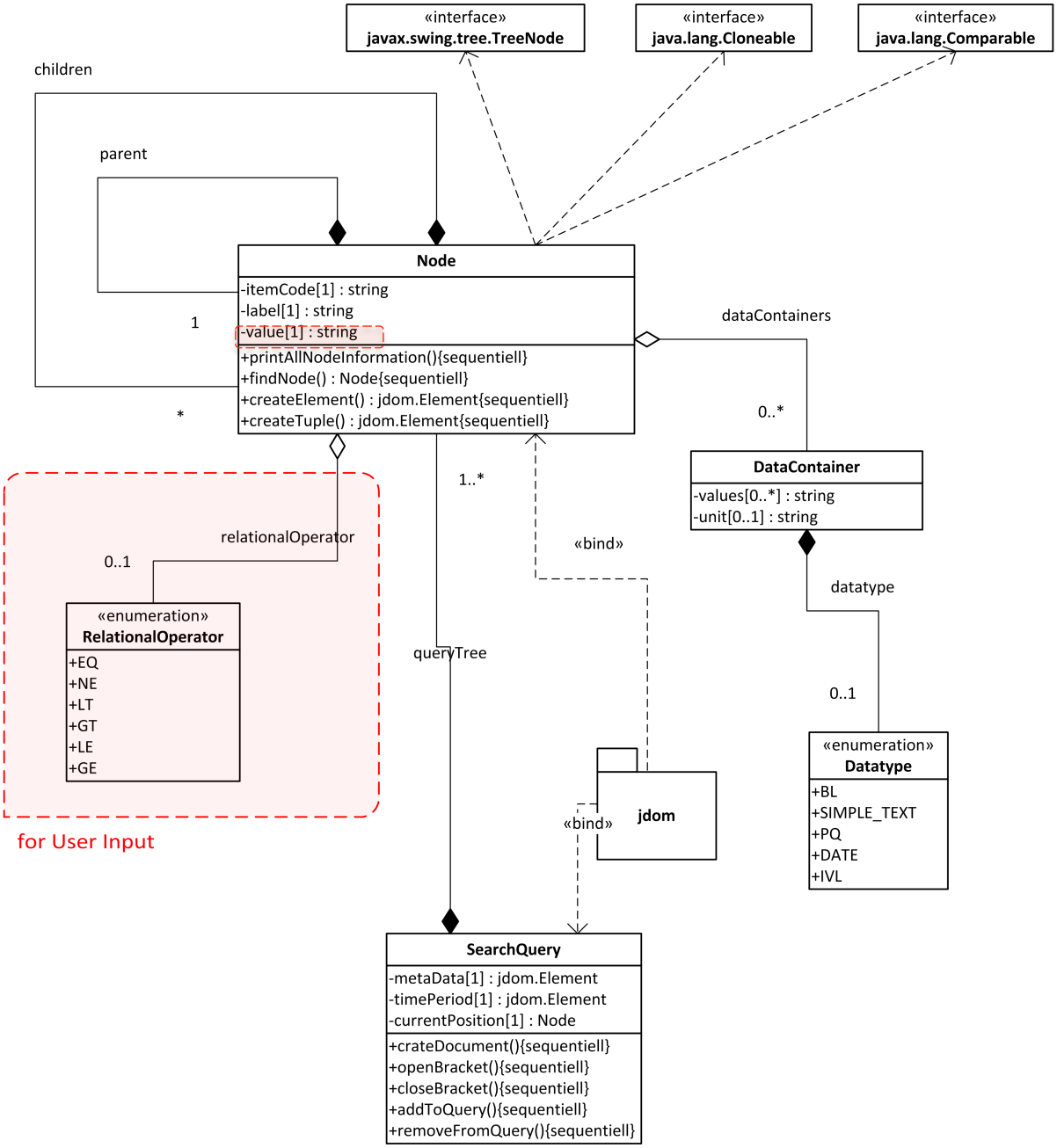

Abbildung 16: Klassendiagramm der Implementierung. Rot markierte Elemente sind für die Speicherung der benutzerdefinierten Werteeinschränkung im Node vorhanden. Getter- und Settermethoden, Konstruktoren bzw. Testmethoden sind in diesem Klassendiagramm nicht enthalten.

4.2.1 Node-Klasse

Die Infoitems können als XML-Format aus dem ATR geladen werden (Abschnitt 3.3.2.1). Diese XML-Information wird im Document Consumer anhand JDOM geparst und in Node-Objekte rekursiv gespeichert. Jedes Node-Objekt bzw. jeder Knoten in der Node-Struktur entspricht dabei einem Infoitem. Es steht somit eine Node-Struktur mit allen Infoitems zur

Verfügung. Für eine Suchanfrage können dann Teile dieser Struktur bzw. einzelne Node-Objekte verwendet werden. Teile der Struktur werden benötigt, damit bei einer Suche eines komplexen Infoitems (welches noch weitere Infoitems beinhalten kann) der komplette Teilbaum der Struktur angezeigt werden kann. Daraus können dann leicht weitere Infoitems ausgewählt werden, ohne nach ihnen suchen zu müssen. Die jeweiligen Teile bzw. Node-Objekte werden dann zum SearchQuery-Objekt (Abschnitt 4.2.3) hinzugefügt.

Jedes Node-Objekt beinhaltet:

- Strukturinformationen
    - Einen Elternknoten
    - Mehrere Kindknoten
- beschreibende Informationen
    - ItemCode: eindeutige Nummer zur Identifizierung des Knotens
    - Label: Bezeichnung des Infoitems
    - DataContainer: Auswahlmöglichkeiten, Einheit, Datentyp, etc.
- Variablen zur Speicherung der Benutzereingabe (bei einem primitiven Infoitem)
    - Value: beinhaltet den benutzerdefinierten Wert der Parametrisierung
    - Operator: definiert die Relation von „Value“ zur Suchanfrage

In der Node-Klasse können außerdem neben den Getter- und Setter-Methoden bzw. Konstruktoren folgende Methoden aufgerufen werden:

- findNode(): findet einen Node im Baum anhand des übergebenen ItemCodes
- createElement(): erstellt ein XML-Element aus einem Datenelement (z.B. ItemCode, Label) eines Knotens mit JDOM
- createTuple(): erstellt ein Tuple aus dem angegebenen Knoten bzw. aus den vorhin generierten XML-Elementen

4.2.2 DataContainer-Klasse

Die DataContainer-Klasse beinhaltet alle Informationen zu einem primitiven Infoitem, welche für die Beschreibung des Datentyps und somit für Werteeinschränkung relevant sind.

Diese Informationen umfassen:

- Datatype: gibt als Enumeration den Datentyp der Instanz an

- Values: Werte, welche vordefiniert sind und zur Parametrisierung ausgewählt werden können
- Unit: Einheit des verwendeten Datentyps

Jedes Infoitem (jeder Knoten in der Node-Struktur) kann mehrere Datentypen besitzen. Für jeden Datentyp existiert eine eigene DataContainer-Instanz. Somit enthält ein Node-Objekt eine Liste an DataContainer-Instanzen (Abbildung 16).

Sowohl bei der Parametrisierung (welche Werte können ausgewählt werden, welche Einheit besitzt der Wert), als auch bei der Erstellung der Query (welche Einheit wird für das Tuple benötigt – Abschnitt 3.3.2.2) wird die DataContainer-Klasse eingebunden.

4.2.3 SearchQuery-Klasse

Mit der SearchQuery-Klasse kann eine neue Suchanfrage nach dem Baukastenkonzept erstellt werden. Die Klasse unterstützt dabei die in Abschnitt 4.1.1 erläuterten Funktionen des Suchanfragen-Editors: Es können beliebige Infoitems in der Node-Struktur gesucht, zur SearchQuery hinzugenommen und parametrisiert werden. Die Infoitems werden mit einem Operator verknüpft und können mit Klammern verschachtelt und strukturiert werden (Abschnitt 3.4.2). Aus einem SearchQuery-Objekt, welches die Infoitems als Node-Objekte (Abschnitt 4.2.1) beinhaltet, kann dann ein valide XML-Suchanfrage erzeugt werden, die an den Document Consumer geschickt und dort ausgeführt wird.

Ein SearchQuery-Objekt beinhaltet:

- Beschreibende Information:
    - metaData: beinhaltet die Metadaten der aktuellen Suche (Abschnitt 3.3.2.2)
    - timePeriod: Zeitbereich, welcher die Dokumente bei der Suche einschränkt.
- Infoitems, nach denen gesucht wird
    - queryTree: eine Node-Struktur, welche alle für die Suche gewünschten Infoitems als Baum-Struktur beinhaltet (Abbildung 16).
    - currentPosition: gibt die aktuelle Position des Zeigers (Curser) an, wo das nächste Infoitem hinzugefügt wird (Abschnitt 3.4.2).

Neben den Getter- und Settermethoden bzw. Konstruktoren existieren in der Klasse außerdem folgende Methoden:

- addToQuery(): fügt ein Infoitem zur aktuellen Suchanfrage an der Position „currentPosition“ hinzu.
- removeFromQuery(): löscht ein ausgewähltes Infoitem aus der Suchanfrage
- openBracket(): erstellt einen Container und setzt den Zeiger „currentPosition“ in den Container. Der Container wird dabei technisch als leerer Knoten in der queryTree-Struktur angelegt.
- closeBracket(): schließt den Container, d.h. der Zeiger „currentPosition“ springt in den Elternknoten des Containers zurück.
- createDocument(): Eine valide XML-Query wird aus dem queryTree anhand JDOM erzeugt. Dazu werden Metadaten und Zeitbereich als XML-Element eingebunden und im queryTree rekursiv die Methode createTuple() (aus der Node-Klasse) aufgerufen.

4.3 Zu Teilziel 3: Test der Implementierung

In diesem Abschnitt werden die Ergebnisse der Modultests beider Schnittstellen und des Integrationstests im fertigen Document Consumer dargestellt (Abschnitt 3.5). Die Tests wurden erfolgreich absolviert, d.h. das Ist-Ergebnis entspricht in allen Fällen (Abschnitt 3.5) dem Soll-Ergebnis, womit auch gleichzeitig die Kommunikation zu den erweiterten Akteuren funktioniert.

4.3.1 Test der ATR Schnittstelle

Die folgenden Abbildungen zeigen das Test-Programm für die ATR Schnittstelle. Wählt man, wie in Abbildung 17 dargestellt, das Infoitem „ItemCode: 39.1 – Hyperglykämie“ aus, dann wird auf der Konsole der entsprechende Inhalt ausgegeben (Abbildung 18). Der Inhalt wird für jeden syntaktischen Fall aus der Spezifikation mit der Ausgabe verglichen, um so die richtige Funktionsweise der Schnittstelle überprüfen zu können. Abbildung 19 zeigt das Infoitem „Körpergewicht“. Hier ist auch der Datentyp „kg“ korrekt in der Datenstruktur vorhanden.

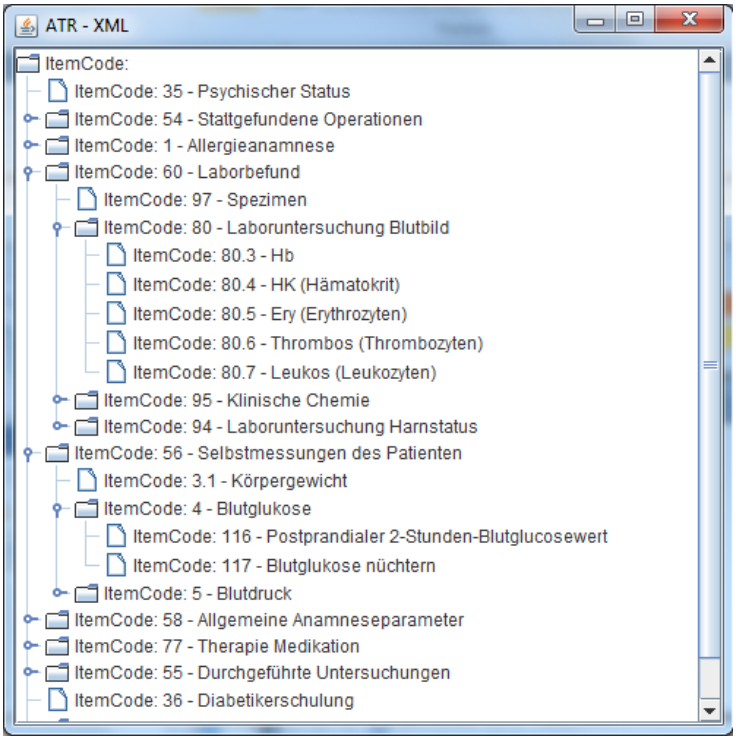
ATR - XML
ItemCode:
ItemCode: 35 - Psychischer Status
ItemCode: 54 - Stattgefundene Operationen
ItemCode: 1 - Allergieanamnese
ItemCode: 60 - Laborbefund
ItemCode: 97 - Spezimen
ItemCode: 80 - Laboruntersuchung Blutbild
ItemCode: 80.3 - Hb
ItemCode: 80.4 - HK (Hämatokrit)
ItemCode: 80.5 - Ery (Erythrozyten)
ItemCode: 80.6 - Thrombos (Thrombozyten)
ItemCode: 80.7 - Leukos (Leukozyten)
ItemCode: 95 - Klinische Chemie
ItemCode: 94 - Laboruntersuchung Harnstatus
ItemCode: 56 - Selbstmessungen des Patienten
ItemCode: 3.1 - Körpergewicht
ItemCode: 4 - Blutglukose
ItemCode: 116 - Postprandialer 2-Stunden-Blutglucosewert
ItemCode: 117 - Blutglukose nüchtern
ItemCode: 5 - Blutdruck
ItemCode: 58 - Allgemeine Anamneseparameter
ItemCode: 77 - Therapie Medikation
ItemCode: 55 - Durchgeführte Untersuchungen
ItemCode: 36 - Diabetikerschulung

Abbildung 17: Grafische Ausgabe des Testprogramms, welches die Infoitems aus dem ATR-XML einliest und in eine Node-Struktur speichert.

```
itemCode=39.1 DE=Hyperglykämie EN=Hyperglycemia

Datacollection Size=1
Datentyp0: SIMPLE_TEXT
Unit des aktuellen Datentyps:

value0: Morgens
value1: Mittags
value2: Vormittags
value3: Abends
value4: Nachmittags
value5: Nachts
```

Abbildung 18: Ausgabe der Informationen zum Infoitem auf der Konsole, nachdem das Infoitem im Testprogramm angeklickt wurde.

```
itemCode=3.1 DE=Körpergewicht EN=body weight

Datacollection Size=1
Datentyp0: PQ
Unit des aktuellen Datentyps: kg
```

Abbildung 19: Ausgabe des Infoitems „Körpergewicht“. Hier ist auch die Einheit „kg“ im DataContainer enthalten.

4.3.2 Test der Document Crawler Schnittstelle

Eine fiktive Query könnte lauten: „liefere mir alle Dokumente der letzten 6 Monate, welche gemessene Blutglukosewerte beinhalten oder glykosyliertes Hämoglobin A1C über 6 mmol/l aufweisen“.

Die Suche beinhaltet neben dem Zeitbereich und den Metadaten noch 2 Infoitems:

- Blutglukose: ItemCode 4 (komplexes Infoitem)
- glykosyliertes Hämoglobin A1C - HbA1c: ItemCode 86.6 (primitives Infoitems)

Folgender Code erstellt die oben genannte Suchanfrage (nach den erläuterten Schritten in Abschnitt 3.5.2):

```
public static void main(String[] args) throws Exception {
        //Verbindung zum ATR aufbauen
        ATRConnector atr = new ATRConnector();

        //Erzeugung des Wurzelknotens des Infoitem-Baum
        Node root = atr.getATRInfo();

        //neue Suchanfrage
        SearchQuery sq = new SearchQuery();
        //Zeitbereich: letzte 6 Monate
        sq.setTimePeriod(new TimePeriod(6));

        Node nodeOfInterest1 = root.findNode("4");
        Node nodeOfInterest2 = root.findNode("86.6");

        //HbA1c parametrisieren "mit größer als 6":
        nodeOfInterest2.setOperator(RelationalOperator.GT);
        nodeOfInterest2.setValue("6");

        //Nodes zur SearchQuery hinzufügen
        sq.addToQuery(nodeOfInterest1);
        sq.addToQuery(nodeOfInterest2);

        //XML Dokument erstellen und ausgeben
        Document docOutput = sq.createDocument();
        XMLOutputter fmt = new XMLOutputter
        ( org.JDOM.output.Format.getPrettyFormat());
        fmt.output( docOutput, System.out );
}
```

Abbildung 20: Main-Methode des Testprogramms für die Suchanfrage „liefere mir alle Dokumente der letzten 6 Monate, welche gemessene Blutglukosewerte beinhalten oder glykosyliertes Hämoglobin A1C über 6 mmol/l aufweisen“.

Nach Ausführen des Programms wird folgende Suchanfrage ausgegeben:

```
<?xml version="1.0" encoding="UTF-8"?>
<SearchQuery>
  <MetaData>
    <author>99999</author>
  </MetaData>
  <TUPLECOLLECTION operator="AND">
    <TUPLECOLLECTION operator="AND">
      <!--Intervall - Collection-->
      <TUPLE>
        <ItemCode>999</ItemCode>
        <Unit />
        <Operator>GT</Operator>
        <Value>2011-06-15T06:09:50</Value>
      </TUPLE>
      <TUPLE>
        <ItemCode>999</ItemCode>
        <Unit />
        <Operator>LT</Operator>
        <Value>2011-12-15T06:09:50</Value>
      </TUPLE>
    </TUPLECOLLECTION>
    <TUPLECOLLECTION operator="AND">
      <TUPLE>
        <!--Blutglukose-->
        <ItemCode>4</ItemCode>
        <Unit />
        <Operator />
        <Value />
      </TUPLE>
      <TUPLE>
        <!--HbA1c (glykosyliertes Hämoglobin A1C)-->
        <ItemCode>86.6</ItemCode>
        <Unit />
        <Operator>GT</Operator>
        <Value>6</Value>
      </TUPLE>
    </TUPLECOLLECTION>
  </TUPLECOLLECTION>
</SearchQuery>
```

Abbildung 21: Ausgabe der XML-Suchanfrage nach Ausführen des Codes aus Abbildung 20

Diese Ausgabe wird dann mit der Schnittstellenspezifikation verglichen. Das oben demonstrierte Beispiel wurde aus der finalen Implementierung erstellt. Der Test war somit erfolgreich. Während des Entwicklungsprozesses diente dieser Test als Überprüfung des Teilergebnisses. Lieferte der Test nicht die gewünschte Ausgabe, musste die Implementierung regressiv geändert werden. Dieser Prozess wurde so lange wiederholt, bis das Ist-Ergebnis dem Soll-Ergebnis entsprach.

Als Antwort des Document Crawlers wird ein XHTML-Dokument retourniert (Abbildung 22 und Abbildung 23). Dieses kann direkt im Front-End des Document Consumers angezeigt werden (Abschnitt 4.1.2), im Test wurde sie jedoch nur ausgegeben.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Suchanfrage: HbA1c (glykosyliertes Hämoglobin A1C) Zeitraum:
     6 Monate</title>
   </head>
   <body>
     <table>
       <thead>
         <tr>
           <th>Kategorie</th>
           <th>Unterkategorie</th>
         </tr>
       </thead>
       <tbody>
         <tr>
           <td>CLUSTER</td>
           <td>Analyse</td>
           <td>
             <value>HbA1c (glykosyliertes Hämoglobin A1C)</value>
           </td>
           <td>
             <value>HbA1c (glykosyliertes Hämoglobin A1C)</value>
           </td>
         </tr>
         <tr>
           <td>CLUSTER</td>
           <td>Ergebnis</td>
           <td>
             <value>10.0 %</value>
           </td>
           <td>
             <value>10.0 %</value>
           </td>
         </tr>
         <tr>
           <td>CLUSTER</td>
           <td>Referenzbereich</td>
           <td>
             <value>4.4 % - 6.0 %</value>
           </td>
           <td>
             <value>4.4 % - 6.0 %</value>
           </td>
         </tr>

      ...
```

Abbildung 22: 1. Teil der XHTML-Antwort des Document Crawlers nachdem die Suche ausgeführt wurde. Zusammen mit dem 2. Teil (Abbildung 23) entspricht es der vollständigen Ergebnis-Rückgabe.

```
        ...
          <tr>
            <td>CLUSTER</td>
            <td>Interpretation</td>
            <td>
              <value>H</value>
            </td>
            <td>
              <value>H</value>
            </td>
          </tr>
          <tr id="footer">
            <td/>
            <td/>
            <td>
              <value>
                <span class="docType">Arztbrief vom Internisten </span>
                <span class="docId">ZKArche.555514.[B@18e9a40110</span>
                <span class="docTime">2011-09-13</span>
              </value>
            </td>
            <td>
              <value>
                <span class="docType">Laborbefund</span>
                <span class="docId"> ZKArche.555514.[B@30e566d73</span>
                <span class="docTime">2011-09-13</span>
              </value>
            </td>
          </tr>
        </tbody>
      </table>
    </body>
</html>
```

Abbildung 23: 2. Teil der XHTML-Antwort des Document Crawlers nachdem die Suche ausgeführt wurde.

4.3.3 Integrationstest im fertigen Document Consumer

Nachdem die Schnittstellen erfolgreich getestet wurden, wurden sie in den fertigen Document Consumer[13] integriert. Im Folgenden wird nun die Funktion für das Ausführen inhaltlicher Suchanfragen und die dafür entwickelten Schnittstellen im fertigen Document Consumer präsentiert. Hier gibt es zwei verschiedene Versionen (Abschnitt 3.3.1):

1. Der Document Consumer unterstützt frei benutzerdefinierte Suche anfragen, welche nach dem Baukastenprinzip im Editor zusammengestellt werden können.
2. Der Document Consumer unterstützt neben den vorgefertigten Suchanfragen und Suchschablonen sogenannte Kurzabfragen. Eine Kurzabfrage ermöglicht es nur nach einem Infoitem zu suchen, welches eventuell[14] parametrisiert werden kann. Diese vereinfachte Version wird im Projekt EHR-Arche für die weiteren Projektschritte verwendet.

Bei der folgenden Erläuterung einer Suche, werden beide Versionen berücksichtigt. Sie unterscheiden sich lediglich in der Auswahl der Suche und dass in Version 2 kein Editor aufgerufen wird.

Der Benutzer startet den Webservice des Document Consumers und loggt sich mit seinen Benutzerdaten ein. Danach erscheint die Patientensuche, in der der Benutzer einen Patienten auswählen kann, zu dem er Dokumente suchen möchte (Abbildung 24). Im Menü unter „Modus“ kann der Benutzer zwischen der Standard-Metadatensuche und der erweiterte archetypenbasierten Suche wählen.

Abbildung 24: Patientensuche im Document Consumer

[13] beim fertigen Document Consumer handelt es sich um einen Prototypen, welcher als Grundlage für die weiteren Projektschritte dienen soll
[14] wenn es sich bei der Auswahl um ein primitives Infoitem handelt

Im Folgenden wird die archetypenbasierte Suche beschrieben. Als nächstes muss die gewünschte Suchart ausgewählt werden. Die möglichen Sucharten unterscheiden sich dabei, je nach o.g. Version (Abbildung 25 und Abbildung 26):

1. Fertige bzw. vordefinierte Suchanfrage: Anfragen die oft verwendet werden und keine Änderungen benötigen. Beispiel: Erstgespräch, Notfall
2. Anpassbare Suche bzw. Suchschablone: Fertige Suchanfragen, bei denen noch Werte parametrisiert werden können
3. Freie bzw. eigene Suche
    a. Version 1: Suchanfrage kann komplett flexibel mit Suchanfragen-Editor erstellt werden.
    b. Version 2: freie Suche beinhaltet nur ein Infoitem, welches eventuell parametrisiert werden kann

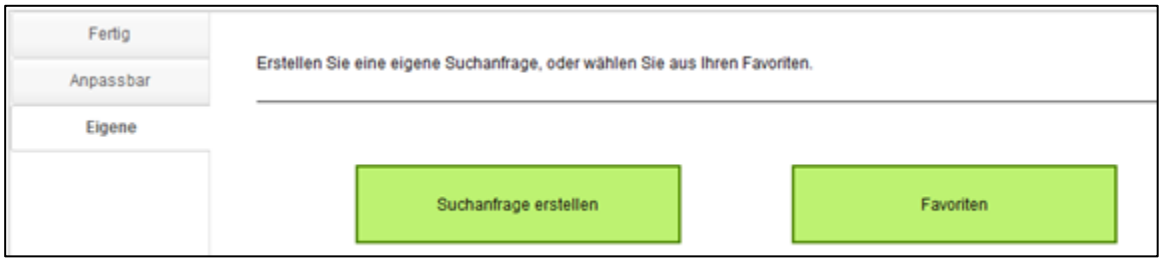

Abbildung 25: Auswahl der Suchart im Document Consumer der Version 1: freie Suchanfragen sind mit Editor beliebig erstellbar

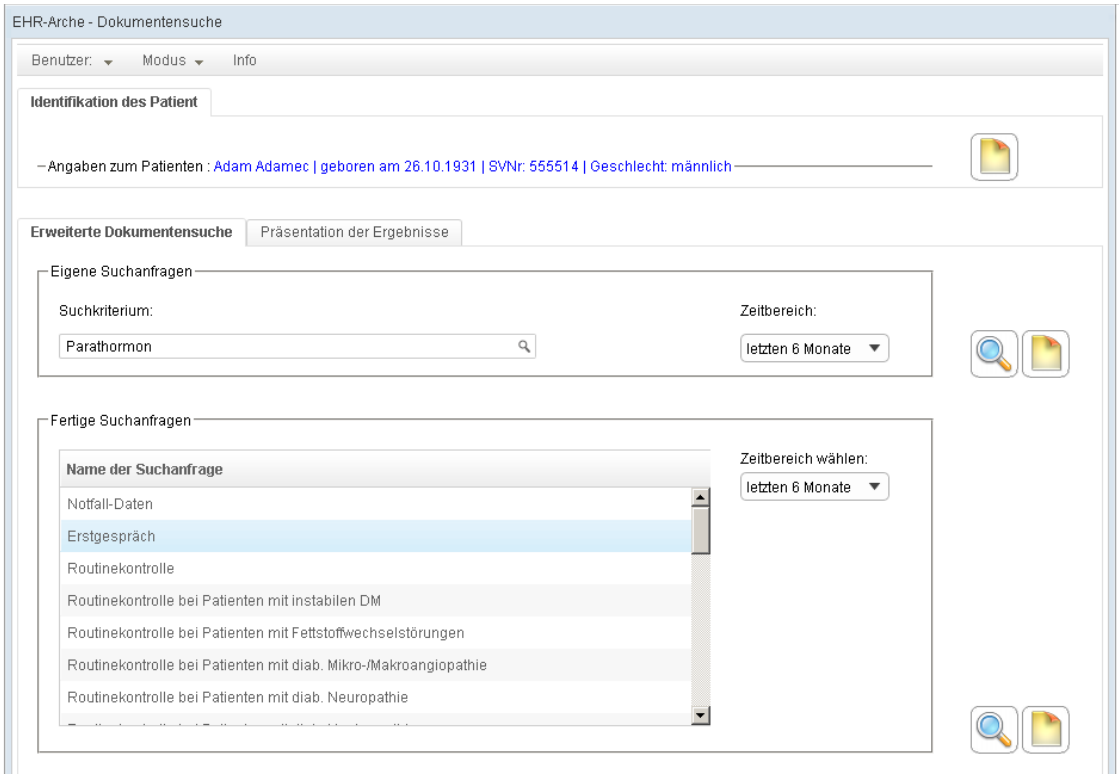

Abbildung 26: Auswahl der Suchart im Document Consumer der Version 2: neben fertigen Suchanfragen kann hier nur nach einem Infoitem gesucht werden → unter „Eigene Suchanfragen“

Hat der Benutzer die eigene Suche aus Version 1 ausgewählt, startet der Suchanfragen-Editor (Abbildung 27). Gleichzeitig verbindet das Back-End mit dem ATR, lädt die Infoitem-XML und parst diese in JAVA-Objekte (Abschnitt 4.1.2). Im Editor findet man die definierten Editorfunktionen aus Abschnitt 4.2.3. Unter „Zeitbereich“ (1)[15] kann der Zeitbereich der Suche eingeschränkt werden, d.h. der Zeitbereich wird in der SearchQuery entsprechend der Eingabe gestellt. Bei „Suchkriterium“ (2) kann nach einem Infoitem gesucht werden. Nachdem „Auswählen“ angeklickt wurde, wird das gesuchte Infoitem inklusive seiner Kinder (bzw. die Baumstruktur ab dem ausgewählten Infoitem) im Anzeigefenster (3) dargestellt. Hier kann man einzelne Elemente des Teilbaums mit einer Checkbox auswählen und eventuell (wenn es ein primitives Infoitem ist) parametrisieren (Abbildung 28). Das Infoitem wird mit „Hinzufügen“ zur Suchanfrage (bzw. im Back-End zur SearchQuery) hinzugefügt, wo es dann in der Vorschau der aktuellen Suchanfrage (6) angezeigt wird. Über die Operatoren (4) und Klammerfunktionen (5) kann die Struktur der ausgewählten Infoitems nach dem Curserprinzip (Abschnitt 3.4.2) festgelegt werden. Hat man die gewünschte Suchanfrage erstellt, kann diese über „Ausführen“ (7) weiterverarbeitet

[15] steht für die Nummerierung in Abbildung 27

werden. Dazu generiert der Document Consumer aus der internen SearchQuery-Struktur (Abschnitt 4.2.3) eine valide XML-Query (Abschnitt 3.3.2.2) und schickt diese an den Document Crawler.

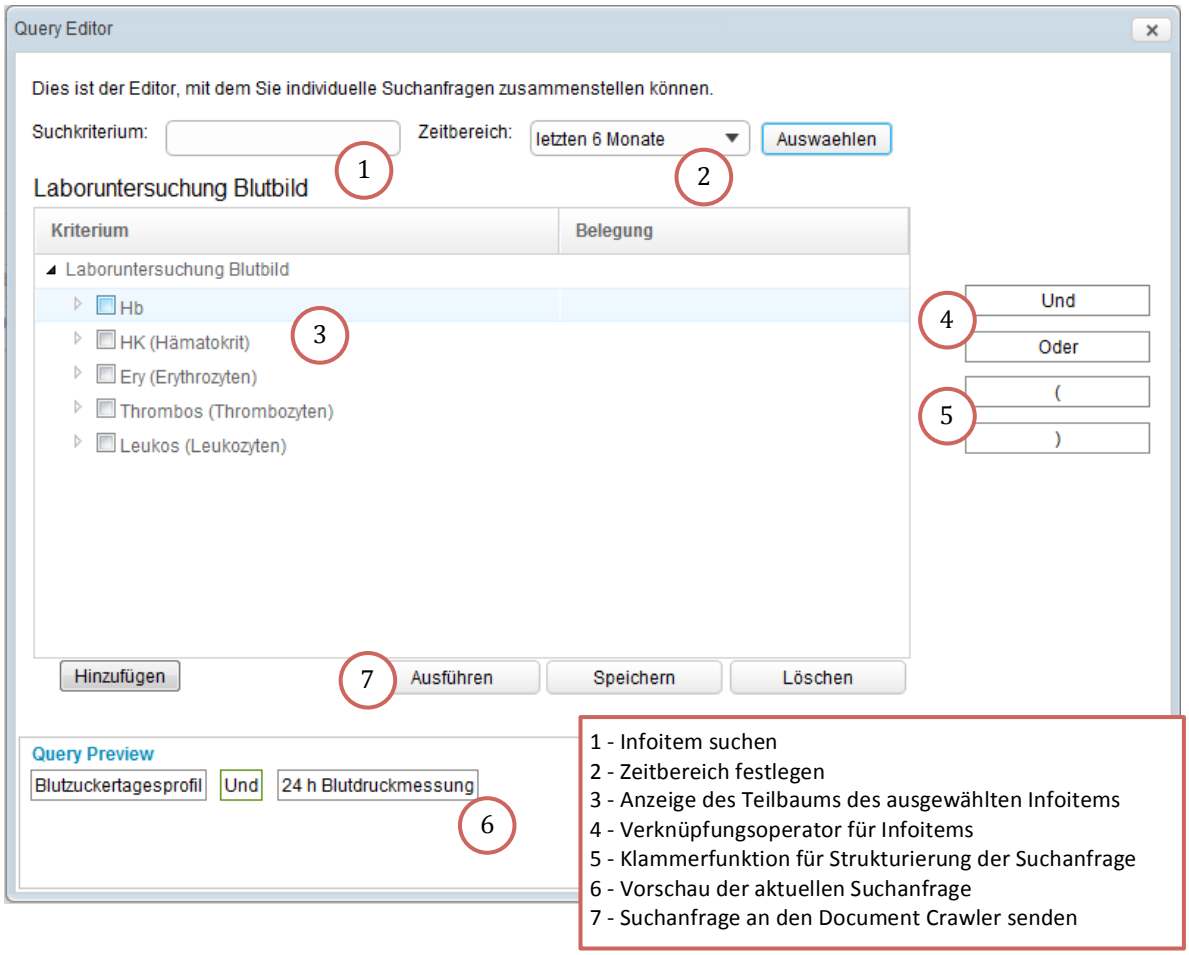

Abbildung 27: Suchanfragen-Editor im Document Consumer mit Bechschreibung zu den Funktionen.

Abbildung 28 beschreibt eine Beispielsuchanfrage, in der bereits nach dem Infoitem „HbA1c“ gesucht wurde. Der Teilbaum, welcher dem Infoitem untergeordnet ist, wird ebenso dargestellt (*Ergebnis, Referenzbereich, Interpretation, Delta*). Über die Checkbox vom jeweiligen Infoitem, könnten untergeordnete Infoitems ebenso ausgewählt werden. Da es sich bei „Ergebnis“ um ein primitives Infoitem handelt, kann dieses genauer parametrisiert werden.

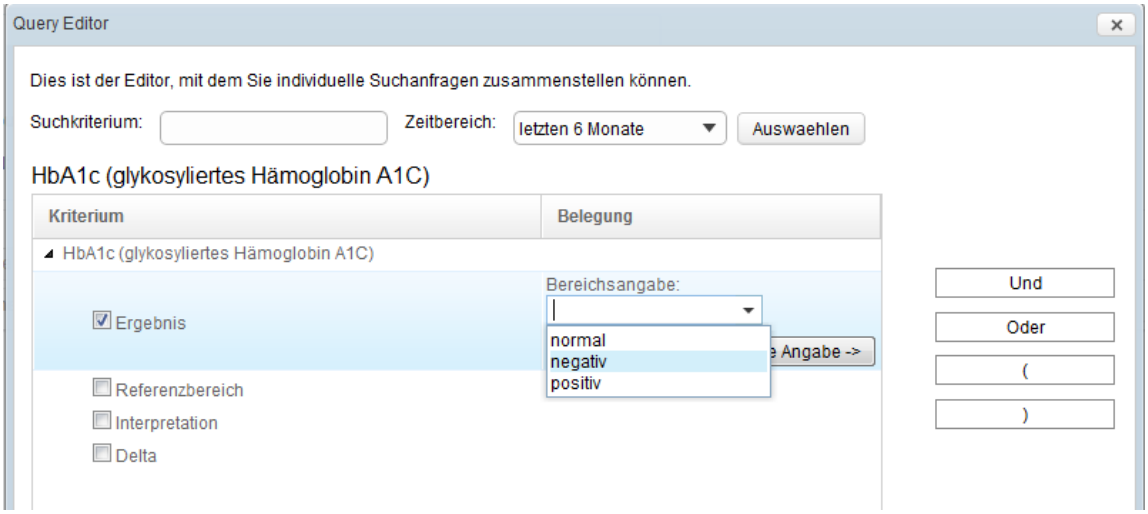

Abbildung 28: Document Consumer mit ausgewähltem Infoitem (*Ergebnis*) und deren Auswahlmöglichkeiten in der Parametrisierung (*normal, negativ, positiv*).

Die folgenden Schritte sind wieder bei beiden Versionen ident. Der Document Crawler wertet die Suche aus und schickt die Ergebnisse als XHTML an den Document Consumer zurück. Dieser kann die Ergebnisse zusammen mit dem Link zum eigentlichen Dokument übersichtlich darstellen (Abbildung 29).

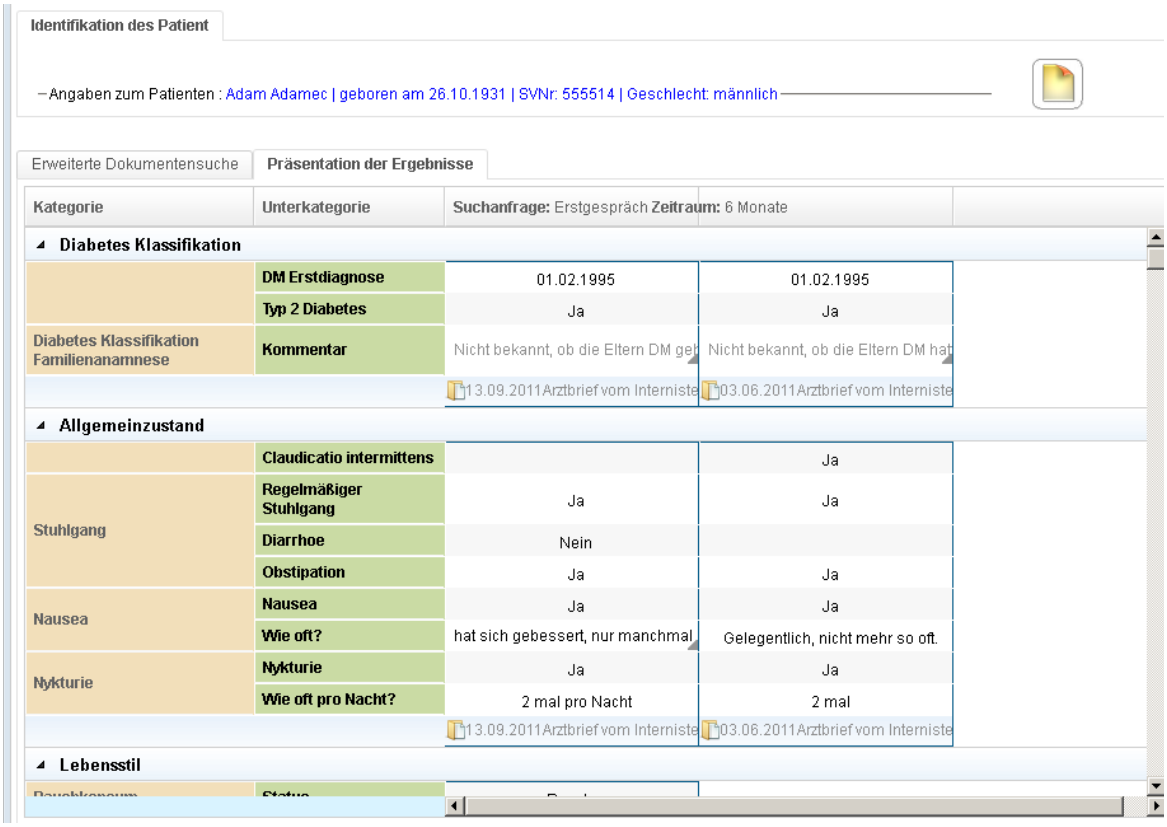

Abbildung 29: Ergebnisrückgabe des Document Crawlers. Hier werden die Infoitems, nach denen gesucht wurde, einzeln aufgelistet, zusammen mit dem jeweiligem Wert und dem Link zum Dokument.

Der Benutzer kann dann das gewünschte Ergebnis anklicken und gelangt direkt zum Ergebnis im jeweiligen Dokument (Abbildung 30).

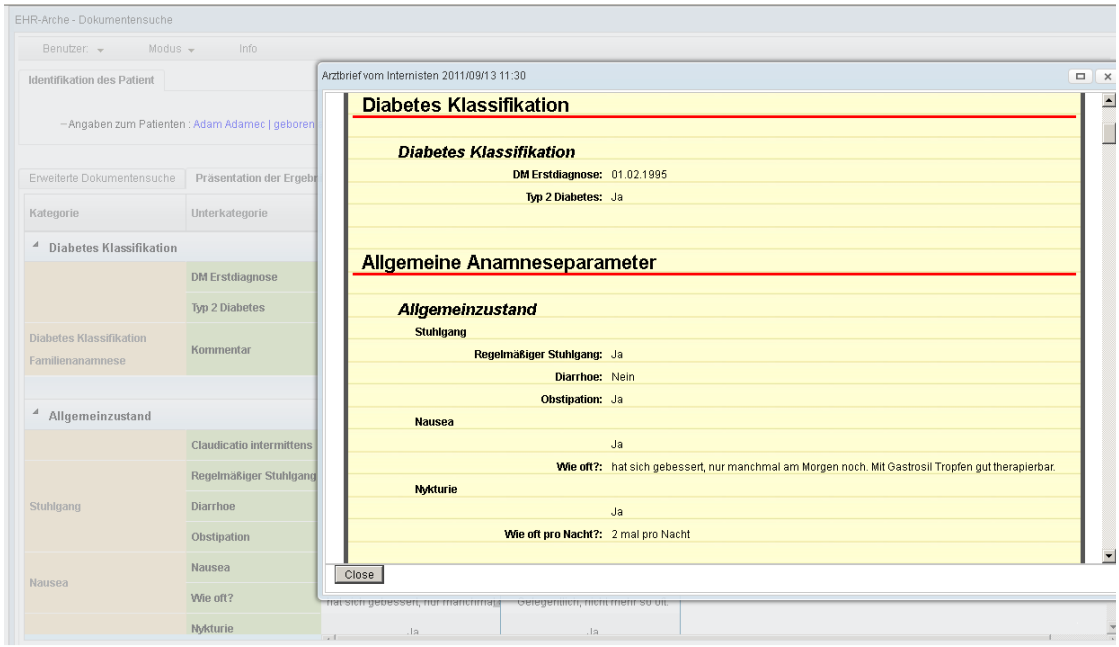

Abbildung 30: Klickt der Benutzer auf eines der Ergebnisse in Abbildung 29, wird das zugrunde liegende Dokument geöffnet und der Bildausschnitt springt genau zu der Stelle, an der das Ergebnis zu finden ist.

5. Diskussion

5.1 Diskussion der Methoden

5.1.1 Vorgehensweise des Entwicklungsprozesses

Der Entwicklungsprozess der Schnittstellen baut auf dem Wasserfallmodell auf. Dieser Ansatz wurde gewählt, um die nötige Flexibilität im Entwicklungsprozess zu integrieren. Da speziell das Konzept für die Erstellung einer flexiblen Suchanfrage ein sehr komplexes Problem darstellt und dies in Zusammenarbeit mit Dr. Samrend Saboor und Dominik Schweiger erstellt wurde, konnte nicht von Anfang an ein klar abgeschlossenes Konzept ohne Versuchsimplementierungen und deren Test erstellt werden. Die strikten Vor- und Zurückkopplungen des Wasserfallmodells schreiben vor, dass nur von einem Projektschritt zum nächsten/vorigen gesprungen werden kann. Es mussten jedoch oft auch noch während der Implementierungs- bzw. Testphase Konzeptänderungen durchgeführt werden, wodurch die strikten Kopplungen des (erweiterten) Wasserfallmodells etwas gelockert werden mussten. Hier wäre eventuell ein evolutionärer Entwicklungsansatz, wie z. B. das Spiral-Modell, besser geeignet.

5.1.2 Auswahl des Parserframeworks

Die Auswahl des JDOM-Parserframeworks erwies sich als gut gewählt: Der Parser konnte nach kurzer Einarbeitungszeit ohne große Probleme als Implementierung umgesetzt werden. Weiters ist die baumorientierte Verarbeitung von JDOM vorteilhaft zum Einlesen bzw. Schreiben der ebenso baumartig strukturierten XML-Nachrichten für die Kommunikation: Sowohl das Parsen des Infoitem-XML, als auch das Serialisieren der XML-Query, verlangen eine flexible Erstellung und Verarbeitung einzelner Knoten bzw. des kompletten Baums. JDOM ermöglicht dies sowohl als XML, als auch als intern als JAVA-Objekte. Auch die Parsgeschwindigkeit von JDOM stellt kein Performanceproblem dar: Sowohl das rekursive Parsen des Infoitem-XML als auch die Erstellung der XML-Suchanfrage läuft nahezu in

Echtzeit[16]. Dies bedeutet jedoch nicht, dass nur JDOM für dieses Problem geeignet ist. JDOM zählt zwar immer noch zu den am meist genutzten XML-Parser-APIs für JAVA [44], es gibt jedoch auch neuere APIs wie XOM, welches zum Beispiel kleiner als JDOM ist und noch einige Verbesserungen wie Performancesteigerung aufweist. Ein bedeutender Grund für die heute immer noch große Beliebtheit des älteren JDOM ist mit Sicherheit die im Internet reichlich verfügbare Dokumentation.

5.2 Diskussion der Ergebnisse

Der Document Consumer ist in der Lage, die benötigten Infoitems aus dem ATR anzufordern, zu parsen und diese für Suchanfragen zu verwenden. Die Struktur zur Erstellung einer Suchanfrage im Suchanfragen-Editor ist vorhanden, welche im Document Consumer Back-End zu einer XML-Suchanfrage transformiert und an den Document Crawler verschickt werden kann. Dort wird die Suchanfrage ausgeführt und die Ergebnisse werden an den Document Consumer retourniert und angezeigt. Die dafür nötigen Schnittstellen wurden in dieser Arbeit konzeptioniert, implementiert und getestet. Durch die entwickelten Schnittstellen wird die Kommunikation zu den neuen Akteuren im erweiterten XDS-Profil ermöglicht.

5.2.1 Zu Teilziel 1: Konzept für die Schnittstellen zu den erweiterten XDS-Akteuren

Das Schnittstellenkonzept baut auf dem Suchanfragen-Editor Konzept und den Schnittstellenspezifikationen der erweiterten XDS-Akteure auf. Das Konzept für die Suchanfragenerstellung deckt alle Anforderungen für die Modellierung einer beliebigen Fragestellung zu einer Suche ab. Dabei können vorgegebene Infoitems vom Benutzer beliebig ausgewählt, verknüpft und parametrisiert werden. Der Vorteil dieser Methode besteht darin, dass aus jeder benutzerdefinierten Suche automatisch eine valide Suchanfrage erzeugt werden kann, da nur die verfügbaren Infoitems in Verbindung mit Containern weiterer Infoitems und Verknüpfungsoperatoren für die Suche verwendet werden. Außerdem können durch diesen generischen Ansatz alle drei Sucharten (vorgefertigte Suchen, Suchschablonen, freie Suchen) mit einem Modell abgedeckt werden: Jede Suchart entspricht einer freien Suche, mit Einschränkungen festgelegter Infoitems (Suchschablonen) oder der zusätzlichen Einschränkung durch Festlegung fixer Parameter (vorgefertigte Suchen). Der Such-

[16] Das Infoitem-XML besteht aktuell mit allen 446 Infoitems und deren zusätzlichen Informationen aus ca. 1363 XML-Elementen.

anfragen-Editor erlaubt somit die flexible Erstellung der Suchanfragen. Es ist jedoch unklar, in wie weit der Benutzer in der Praxis sich selbst Suchanfragen zusammenstellen möchte bzw. ob er nicht eher auf die vorgefertigten Suchen bzw. Suchschablonen zurückgreift. Die Erstellung einer freien Suche ist mit mehr Zeit verbunden, was unter Umständen zu einer Inakzeptanz führen kann. Speziell zu Beginn verlangt es einige Übung, bis eine freie Suchanfrage schnell erstellt werden kann.

Das Schnittstellenkonzept erlaubt durch seine flexible Kommunikation über proprietäre XML-Nachrichten eine einfache Anbindung an andere Akteure. Möchte ein neuer ATR-Akteur (z.B. mit Archetypen zur Krankheit Leukämie) an den Document Consumer binden, muss dieser nur eine der Spezifikation entsprechenden XML-Nachricht erzeugen können. Dasselbe gilt für den Document Crawler Akteur: werden mehrere Crawler-Akteure in einem System integriert (z.B. pro Repository ein Crawler), muss der jeweilige Crawler nur eine valide (der Spezifikation entsprechende) XML-Query empfangen können. Die Schnittstellen binden somit nicht fix an die Akteure, sondern können flexibel verwendet werden.

5.2.2 Zu Teilziel 2: Implementierung der Schnittstellen

Allgemein werden intern im Document Consumer ausschließlich JAVA-Objekte für die Modellierung bzw. Verarbeitung einer Suchanfrage verwendet. Somit müssen intern keine komplizierten Transformationen aus XML-Daten stattfinden. Das Einlesen der Infoitem-XML wird beim Start einmal durchgeführt, anschließend wird nur mehr mit der erstellten Node-Struktur gearbeitet.

Die Implementierung der ATR Schnittstelle (Node-Klasse) ist in der Lage, alle Infoitems als strukturiertes XML-Dokument aus einem ATR anzufordern und diese in eine Node-Struktur zu parsen. Der rekursive Ansatz beim Einlesen der Infoitem-XML erlaubt eine dynamischere Erstellung der internen Objekte. Zum Parsen hätte auch ein reiner DOM-Parser verwendet werden können, welcher die komplette Node-Struktur in einem Schritt als Baumstruktur integriert. Mit dem sequentiell rekursivem Parsen kann jedoch schon beim Einlesen entschieden werden, wie die Daten in den Objekten differenziert gespeichert werden sollen. So können zum Beispiel die Datentypen eines Infoitems schon beim Einlesen in ein DataContainer-Objekt verpackt werden, um die Information später effizienter nutzen zu können. Diese effiziente Speicherung wird bei der Erstellung einer Suchanfrage ausgenutzt. Da eine Suchanfrage im Grunde aus „Teilbäumen“ der ursprünglichen Node-Struktur besteht, beinhalten die Objekte bereits einen Großteil der nötigen Informationen für der Suche. Auswahlmöglichkeiten, Datentypen, Einheiten, etc. sind somit schon

aus dem ATR direkt im Node-Objekt vorhanden und können für die Parametrisierung eines Infoitems übernommen werden. Für die Suchanfrage muss dann nur noch die Verknüpfung der gewählten Infoitems untereinander und die Werteeinschränkung einzelner Infoitems vom Benutzer definiert werden. Die gesammelt (in jedem Node-Objekt) gespeicherte Information erlaubt dann eine einfache Erstellung der XML-Suchanfrage: Da aus jedem Node-Objekt in der SearchQuery ein „TUPLE“ in der XML-Ausgabe erstellt wird, kann JDOM alle nötigen Variablen (ItemCode, Unit, Operator, Value) für die Erstellung ohne zusätzlichen Aufwand direkt aus dem jeweiligen Node-Objekt beziehen.

Eine Schwäche der ATR Schnittstelle könnte die Inflexibilität in Betracht auf Erweiterungen bzw. Änderungen des ATR sein. Wird die Struktur des Infoitem-XML aus dem ATR erweitert oder geändert, so muss auch die Schnittstelle im Document Crawler diesen Änderungen angepasst werden. Entscheidet man sich zum Beispiel neben Deutsch und Englisch eine weitere Sprache zu integrieren und fügt diese als weiteres Sprachenlabel im ATR ein, so muss die Schnittstelle so umgebaut werden, dass auch die neue Sprache in der Node-Struktur eingelesen und gespeichert wird.

Da die Implementierung der SearchQuery-Klasse auf dem Suchanfragen-Editor aufbaut, lassen sich die Vor- und Nachteile des Editors auch in der Implementierung im Back-End erkennen. Die bisherige Lösung des Suchanfragen-Editors ist nicht sehr benutzerfreundlich: Die Erstellung einer Suchanfrage sollte für den Benutzer flexibel sein. Die Implementierung erlaubt jedoch durch den „Cursor“ nur das Einfügen eines Elementes an einer Position, d.h. der Benutzer kann nicht frei wählen, wo in der Suchanfrage das nächste Infoitem hinzugefügt wird. Eine Lösung wäre hier eine benutzerfreundliche grafische Bedienoberfläche, welche das Editorkonzept besser umsetzt. In der aktuellen Umsetzung des Editors fehlen weiters grundlegende Funktionen, um damit in der Praxis arbeiten zu können. Zum Beispiel das Löschen eines geklammerten Bereichs (einer Ebene) ist im aktuellen Prototypen noch nicht möglich. Aus diesen Gründen wurde die Editorfunktion für den folgenden Projektschritt (Evaluation der Implementierung) auf eine Funktion reduziert: Der Benutzer des Document Consumers kann in freien Suche nur nach einem beliebigen Infoitem suchen, welches er jedoch parametrisieren kann (Kurzabfrage).

Die SearchQuery-Klasse baut dennoch auf dem Editorkonzept auf und unterstützt eine frei erstellte Suche auch weiterhin. Die interne Speicherung der XML-Information in Node-Objekten erlaubt eine flexible bzw. programmiertechnisch einfache Verwendung für eine Suchanfrage. Diese Kriterien sind wichtig für die Erstellung einer freien Suche: Eine kom-

plexe (freie) Suchanfrage wird in der Regel nicht in einem Zug erstellt, sondern es werden Infoitems hinzugefügt, parametrisiert, wieder entfernt und/oder eventuell an einer anderen Stelle eingefügt. Die Implementierung regelt diese Funktion über den „Cursor“. Der Nachteil dieses Prinzips ist die o.g. nicht sehr gute Benutzerfreundlichkeit. Es wurde hier ein Kompromiss gewählt, der sowohl eine einfache Implementierung erlaubt, als auch die gewünschte Funktion gewährleistet. Da es mit diesem Ansatz möglich ist, beliebige Infoitems beliebig zu kombinieren bzw. zu strukturieren, wurde das Ziel mit einer gewissen Reduzierung der Benutzerfreundlichkeit erreicht.

5.2.3 Zu Teilziel 3: Test der Implementierung

In den Modultests wurden zuerst die Schnittstellen einzeln auf ihre Funktion getestet. Jede Schnittstelle entspricht dabei einem eigenen Modul. Anschließend wurden die Schnittstellen im fertigen Document Consumer integriert und getestet (Integrationstest). Sowohl in den Modultests als auch im Integrationstest wurden ausschließlich die funktionalen Anforderungen der Schnittstellen anhand der Schnittstellenspezifikationen getestet. Funktionale Anforderungen beschreiben dabei die Funktionen, welche benötigt werden, damit das System wie gewünscht funktioniert, während nichtfunktionale Anforderungen Qualitätseigenschaften, wie z.B. Benutzerfreundlichkeit, darstellen. Die Resultate beider Schnittstellen konnten leicht überprüft werden (abzählbar unterscheidbare Fälle für den Pars-Algorithmus), daher wurden diese manuell mit einem Testprogramm überprüft. Diese Methode eignete sich sehr gut, da auch nachträgliche Änderungen in der Implementierung einfach und schnell mit dem selben Test-Programm überprüft werden konnten. Da es sich beim Document Consumer um einen Prototypen handelt, der dazu dient, die weiteren Projektschritte durchführen zu können, wurden keine Tests für nicht-funktionale Anforderungen durchgeführt.

In dieser Arbeit wurden außerdem keine Tests mit dem XHTML-Ergebnis des Document Crawlers durchgeführt. Der Test der Schnittstelle zum Document Crawler wurde als erfolgreich angesehen, wenn die erstellte XML-Query den Spezifikationen entsprach.

5.3 Bedeutung der Ergebnisse

Die Systeme des erweiterten XDS-Profils sollen in keiner Weise die Standard XDS-Strukturen beeinflussen bzw. ändern. Somit ist es für die erweiterte archetypenbasierte Suche nötig, den ATR-Akteur und Document Crawler-Akteur vom Document Consumer-Akteur zu differenzieren und flexibel miteinander zu verbinden. Die Verbindung benötigt

die Schnittstellen, welche in dieser Arbeit geplant, erstellt und getestet wurden. Die XDS-Erweiterungen sind somit erst durch die Schnittstellen in der Lage, mit dem Document Consumer zu kommunizieren. Der Document Consumer des erweiterten XDS-Profils ist als Prototyp funktionsbereit und kann in weiterer Folge für den nächsten Projektschritt, der Evaluation der inhaltlichen Dokumentensuche, verwendet werden.

6. Ausblick

Die flexible Editorfunktion ist im fertigen Document Consumer, wegen den in der Diskussion beschriebenen Usability-Gründen, nicht freigeschalten und somit für die Evaluation im Projekt EHR-Arche nicht verfügbar. Im Projekt können nur Kurzabfragen anhand eines benutzerdefinierten Infoitems erfolgen. In Folgeprojekten könnte der Prototyp des Suchanfragen-Editors verbessert und praxistauglich gemacht werden. Dafür müsste die Editorfunktion über eine benutzerfreundlichere Bedienung verfügen. Die Schnittstellen sind zwar vom Editor relativ unabhängig, bei einer umfangreicheren Änderung im Konzept der Suchanfragenerstellung müssten jedoch auch die Schnittstellen angepasst werden. Allgemein handelt es sich bei dem Document Consumer um einen Prototypen, welcher als Grundlage der weiteren Projektschritte dienen soll. Der Consumer wurde nur auf funktionale Anforderungen getestet. Hier sind noch Tests der nicht-funktionalen Anforderungen, wie Benutzerfreundlichkeit, Leistung und Effizient, bei der Erstellung einer Suchanfrage erforderlich.

7. Danksagung

Ich möchte mich hiermit bei allen Projektbeteiligten bedanken, die jederzeit für mich erreichbar waren und mich sehr hilfsbereit unterstützt haben. Ein besonderer Dank gebührt an dieser Stelle meinem Mitstudenten Dominik Schweiger für gute Zusammenarbeit im Projekt.

Den größten Dank möchte ich meinem Betreuer Samrend Saboor aussprechen, der mich beim Arbeiten am Projekt bzw. beim Schreiben der Arbeit mit größter Kompetenz, Sorgfalt und Genauigkeit betreute. Außerdem habe ich durch die zahlreichen konstruktiven Gespräche weitaus mehr gelernt, als nur eine Abschlussarbeit zu verfassen. Vielen Dank!

Zu guter Letzt möchte ich mich bei meinen Eltern bedanken, die mir das Studium ermöglichten, und bei allen Personen, die mich während dem Schreiben dieser Arbeit mit konstruktiven Beiträgen und mit ermunternden Worten unterstützt haben.

Literaturverzeichnis

1. Leiner F, Gaus W, Haux R, Knaup-Gregori P, Pfeifer K. Medizinische Dokumentation: Grundlagen einer qualitätsgesicherten integrierten Krankenversorgung. Lehrbuch und Leitfaden. Stuttgart: Schattauer; 2006.

2. IBM. Machbarkeitsstudie betreffend Einführung der elektronischen Gesundheitsakte (ELGA) im österreichischen Gesundheitswesen Endbericht. 2006.

3. Hackl WO, Hoerbst A, Ammenwerth E. "Why the hell do we need electronic health records?". EHR acceptance among physicians in private practice in Austria: a qualitative study. Methods Inf Med. 2011;50(1):53-61. Epub 2010/11/09.

4. IHE. IT Infrastructure Technical Framework. vol 1 (ITI TF-1, chapter 10), vol 2 (ITI TF-2, chapter 314 and Appendix L). 2007.

5. Basch P. Data Excess and Document Overload: Barriers and Disincentives to an Interconnectes/Interoperable Healthcare System. In: The Data Standards Working Group Report and Recommendations. Connecting for Health, Markle Foundation. 2003:89-96.

6. Hackl WO. Die Elektronische Gesundheitsakte Österreich: Ängste, Befürchtungen und Widerstände aus ärztlicher Sicht. Hall i. Tirol: UMIT; 2008.

7. Hübner-Bloder G, Duftschmid G, Kohler M, C. R, Saboor S, Ammenwerth E. Project EHR -ARCHE - Clinical information needs and archetype-based electronic health records. 2010.

8. Hübner-Bloder G, Duftschmid G, Kohler M, C. R, Saboor S, Ammenwerth E. Clinical situations and information needs of physicians during treatment of Diabetes mellitus patients: a Triangulation Study. 2011. MIE.

9. ISO. ISO 13606-1 - Health informatics - Electronic health record communication. 2008.

10. Dolin RH, Alschuler L, Boyer S, Beebe C, Behlen FM, Biron PV, et al. HL7 Clinical Document Architecture Release 2. Journal of the American Medical Informatics Association. 2006;13(1):30-9.

11. The openEHR foundation. Letzter Zugriff: 14.11.2011; URL: http://www.openehr.org.

12. Haas P. Gesundheitstelematik - Grundlagen Anwendungen Potenziale. Berlin: Springer-Verlag; 2006.

13. Wozak F, Schabetsberger T, Ammmenwerth E. End-to-end security in telemedical networks - a practical guideline. International Journal of Medical Informatics. 2007;76(5-6):484-90.

14. Blobel BG, Engel K, Pharow P. Semantic interoperability - HL7 Version 3 compared to advanced architecture standards. Methods of information in medicine. 2006;45(4):343-53.

15. Dorda W, Duftschmid G, Gerhold L, Gall W, Gambal J. Austria's path toward nationwide electronic health records. Methods of information in medicine. 2008;47(2):117-23. Epub 2008/03/14.

16. epSOS – the European eHealth Project. 2011; Letzter Zugriff: 23.11.2011; URL: http://www.epsos.eu/.

17. Integrating the Healthcare Enterprise Website. Letzter Zugriff: 07.09.2011; URL: http://www.ihe.net/.

18. Rinner C. Electronic health records (EHRs): Data export of health information systems based on the Entity-Attribute-Value model as CEN prEN 13606 compliant EHR extracts by means of Archetypes. Wien: Medizinische Universität Wien; 2007.

19. IHTSDO. SNOMED CT. International Health Terminology Standards Developement Organisation.

20. Beale T. Archetypes and the EHR. Studies in health technology and informatics. 2003;96:238-44. Epub 2004/04/06.

21. ISO. ISO 8879:1986 - Standard Generalized Markup Language (SGML). 1986.

22. W3C. HTML 4.01 Specification. 1999.

23. W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition). 2008.

24. Stock W. Information Retrieval - Informationen suchen und finden. München: Oldenbourg Wissenschaftsverlag GmbH; 2007.

25. Menche N. Biologie, Anatomie, Physiologie. München: Urban & Fischer; 2007.

26. Österreichische Diabetes Gesellschaft. 2011; Letzter Zugriff: 09.09.2011; URL: http://www.oedg.org/.

27. Hübner-Bloder G, Duftschmid G, Kohler M, Rinner C, Saboor S, Ammenwerth E. Systematische Erhebung der Informationsbedürfnisse von Ärzten bei der Behandlung von Diabetes mellitus Patienten. 55 GMDS-Jahrestagung. 2010.

28. Rinner C, Kohler M, Hubner-Bloder G, Saboor S, Ammenwerth E, Duftschmid G. Creating ISO/EN 13606 archetypes based on clinical information needs. Studies in health technology and informatics. 2011;165:43-8. Epub 2011/06/21.

29. Balzert H. Lehrbuch der Software- Technik: Software-Entwicklung. Heidelberg: Spektrum Akademischer Verlag; 2001.

30. Grechenig T, Bernhart M. Softwaretechnik: Mit Fallbeispielen aus realen Entwicklungsprojekten. München: Pearson Studium; 2009.

31. Scholz M, Niedermeier S. JAVA und XML. Bonn: Galileo Press; 2009.

32. W3C. Document Object Model (DOM) Level 3 Core Specification. 2004.

33. dom4j. 2005; Letzter Zugriff: 03.12.2011; URL: http://dom4j.sourceforge.net.

34. Harold ER. XOM. 2002; Letzter Zugriff: 03.12.2011; URL: http://www.xom.nu.

35. McLaughlin B, Hunter J. JDOM. Letzter Zugriff: 01.12.2011; URL: http://www.jdom.org/.

36. Megginson D. SAX. 1998; Letzter Zugriff: 01.12.2011; URL: http://www.saxproject.org.

37. Gamma E, Helm R, Johnson R, Vlissides J. Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software. München: Addison-Wesley Verlag; 2004.

38. Fry C, Slominski A. StAX Reference Implementation. 2006; Letzter Zugriff: 10.12.2011; URL: http://stax.codehaus.org/.

39. Java API for XML Bindung - JAXB. Letzter Zugriff: 14.11.2011; URL: http://jaxb.java.net/.

40. ZKoss Framework - an open-soruce Ajax Web application framework. Potix Corporation; 2011; Letzter Zugriff: 19.10.2011; URL: http://www.zkoss.org/.

41. Foundation TE. Eclipse. Letzter Zugriff: 15.11.2011; URL: http://www.eclipse.org.

42. Subclipse Plugin. Letzter Zugriff: 14.11.2011; URL: http://subclipse.tigris.org.

43. Liggesmeyer P. Software-Qualität: Testen, Analysieren und Verifizieren von Software. Heildelberg: Spektrum Akademischer Verlag; 2009.

44. Poll Results: Which XML object model do you use most? Letzter Zugriff: 03.01.2012; URL: http://www.servlets.com/polls/results.tea?name=doms.

Abbildungsverzeichnis

Abbildung 1: Ablauf des Entwicklungsprozesses eines Technical Frameworks [17]. 6
Abbildung 2: Darstellung des IHE XDS-Profils mit allen beteiligten Akteuren und Transaktionen [4]. 8
Abbildung 3: Ein wohlgeformtes XML-Dokument 12
Abbildung 4: Architektur des erweiterten XDS-Profils. Das Standard XDS-Profil wird um einen Archetype Repository Akteur (für die Aufbewahrung der Archetypen) und einem Document Crawler Akteur (für die Verarbeitung inhaltlicher Suchanfragen) erweitert und mit den Standard-Akteuren verbunden. 18
Abbildung 5: Infoitem 6.3 (Typ 1A Diabetes) kommt im Archetyp „diabetes_classification“ vor, also müssen nur die Dokumenttypen durchsucht werden, in denen dieser Archetyp vorkommt → entspricht allen COMPOSITION Archetypen die in der Grafik angezeigt werden. 20
Abbildung 6: Erweitertes Wasserfallmodell angepasst an die Problemstellung dieser Arbeit 21
Abbildung 7: Vom Projektteam erstelltes Anwendungsfalldiagramm mit Funktionen der erweiterten XDS-Akteure.....28
Abbildung 8: Ausschnitt aus dem Infoitem-XML des ATR mit komplexen (z.B. Laborbefund) und primitiven Infoitems (z.B. Ergebnis) 31
Abbildung 9: Beispiel einer validen XML-Query. Es wird nach Dokumenten zwischen dem 07.03.2011 und dem 07.09.2011 gesucht, in denen ein nicht näher definierter „Kalium“-Messwert vorkommt 34
Abbildung 10: Ausschnitt aus dem Infoitem-XML 37
Abbildung 11: Generierte Baumstruktur nach dem Parsen des Infoitem-XML-Ausschnitts aus Abbildung 10 38
Abbildung 12: Beispiel einer Suchanfrage und wie diese theoretisch in der SearchQuery-Klasse aufgebaut wird. Jeder Ebene werden die Infoitems (rot) mit einem benutzerdefinierten Operator verknüpft. Mit der openBracket() Methode kann eine neue Ebene mit eigenem Verknüpfungsoperator erstellt werden.....40
Abbildung 13: Ablauf des Programms zum Testen der Document Crawler Schnittstelle. So kann eine beliebige Suchanfrage mit beliebigen Infoitems erstellt werden. 43
Abbildung 14: Anwendungsfalldiagramm, welches die Use Cases des Suchanfragen-Editors darstellt. ...45
Abbildung 15: Aufbau der Erweiterung des Document Consumers für die Erstellung inhaltlicher Suchanfragen mit den Schnittstellen und deren Verbindung zum ATR und Document Crawler. Wird eine neue Suchanfrage im Suchanfragen-Editor erstellt, werden alle Infoitems aus dem ATR geladen und in eine JAVA-Struktur geparst. Nachdem der Benutzer die Suchanfrage erstellt hat, wird die JAVA-Struktur in eine XML-Query serialisiert und an den Document Crawler geschickt. Die Ergebnisse werden anschließend als XHTML retourniert und direkt im Front-End präsentiert.....47

Abbildung 16: Klassendiagramm der Implementierung. Rot markierte Elemente sind für die Speicherung der benutzerdefinierten Werteeinschränkung im Node vorhanden. Getter- und Settermethoden, Konstruktoren bzw. Testmethoden sind in diesem Klassendiagramm nicht enthalten. 49
Abbildung 17: Grafische Ausgabe des Testprogramms, welches die Infoitems aus dem ATR-XML einliest und in eine Node-Struktur speichert. 53
Abbildung 18: Ausgabe der Informationen zum Infoitem auf der Konsole, nachdem das Infoitem im Testprogramm angeklickt wurde. 53
Abbildung 19: Ausgabe des Infoitems „Körpergewicht“. Hier ist auch die Einheit „kg“ im DataContainer enthalten. 53
Abbildung 20: Main-Methode des Testprogramms für die Suchanfrage „liefere mir alle Dokumente der letzten 6 Monate, welche gemessene Blutglukosewerte beinhalten oder glykosyliertes Hämoglobin A1C über 6 mmol/l aufweisen“. 54
Abbildung 21: Ausgabe der XML-Suchanfrage nach Ausführen des Codes aus Abbildung 20 55
Abbildung 22: 1. Teil der XHTML-Antwort des Document Crawlers nachdem die Suche ausgeführt wurde. Zusammen mit dem 2. Teil (Abbildung 23) entspricht es der vollständigen Ergebnis-Rückgabe. 56
Abbildung 23: 2. Teil der XHTML-Antwort des Document Crawlers nachdem die Suche ausgeführt wurde. 57
Abbildung 24: Patientensuche im Document Consumer 58
Abbildung 25: Auswahl der Suchart im Document Consumer der Version 1: freie Suchanfragen sind mit Editor beliebig erstellbar 59
Abbildung 26: Auswahl der Suchart im Document Consumer der Version 2: neben fertigen Suchanfragen kann hier nur nach einem Infoitem gesucht werden → unter „Eigene Suchanfragen“ 60
Abbildung 27: Suchanfragen-Editor im Document Consumer mit Bechschreibung zu den Funktionen. ...61
Abbildung 28: Document Consumer mit ausgewähltem Infoitem (*Ergebnis*) und deren Auswahlmöglichkeiten in der Parametrisierung (*normal, negativ, positiv*). 62
Abbildung 29: Ergebnisrückgabe des Document Crawlers. Hier werden die Infoitems, nach denen gesucht wurde, einzeln aufgelistet, zusammen mit dem jeweiligem Wert und dem Link zum Dokument. 62
Abbildung 30: Klickt der Benutzer auf eines der Ergebnisse in Abbildung 29, wird das zugrunde liegende Dokument geöffnet und der Bildausschnitt springt genau zu der Stelle, an der das Ergebnis zu finden ist. 63

Tabellenverzeichnis

Tabelle 1: Kontingenztabelle für das Wiederfinden von Information in einer Menge. 14
Tabelle 2: Beispiel einer Kontingenztabelle von einer Dokumentensuche 14
Tabelle 3: Vergleich von Metadatensuche und Archetypensuche 17
Tabelle 4: Bewertungstabelle für die Merkmale der Parser-Frameworks 26
Tabelle 5: Vergleich der verschiedenen XML-Parsertypen 26
Tabelle 6: Fragestellungen mit der zugehörigen Query-Semantik einer Suchanfrage 33

Eidesstattliche Erklärung

Hiermit versichere ich, Marco Schweitzer, geboren am 12.12.1988 in Hall i. Tirol, dass die vorliegende Bachelorarbeit von mir selbständig verfasst wurde. Zur Erstellung wurden von mir keine anderen, als die angegebenen Quellen und Hilfsmittel verwendet.

Hall i. Tirol, 28.02.2012

(Marco Schweitzer)